Ordonnancement sur machines parallèles appliqué à la fabrication de semi-conducteurs : ateliers de photolithographie

Abdoul Bitar

NNT : **2015 EMSE 0808**

THÈSE

présentée par

Abdoul BITAR

pour obtenir le grade de
Docteur de l'École Nationale Supérieure des Mines de Saint-Étienne

Spécialité : Génie Industriel

ORDONNANCEMENT SUR MACHINES PARALLÈLES APPLIQUÉ À LA FABRICATION DE SEMI-CONDUCTEURS : ATELIERS DE PHOTOLITHOGRAPHIE

soutenue à Gardanne, le 11/12/2015

Membres du jury

Président :	Jacques CARLIER	Professeur des Universités, Université de Technologie de Compiègne
Rapporteurs :	Safia KEDAD-SIDHOUM	Maître de Conférences, Université Pierre et Marie Curie, Paris VI
	Ameur SOUKHAL	Professeur des Universités, Université de Tours
Examinateur(s) :	Jacques CARLIER	Professeur des Universités, Université de Technologie de Compiègne
	Lyes BENYOUCEF	Professeur des Universités, Université Aix-Marseille
	Jacques PINATON	Ingénieur, STMicroelectronics, Rousset
Directeur(s) de thèse :	Stéphane DAUZERE-PERES	Professeur, EMSE, Gardanne
	Claude YUGMA	Chargé de Recherche, EMSE, Gardanne
Invité:	Renaud ROUSSEL	Ingénieur, STMicroelectronics, Crolles

Spécialités doctorales :	Responsables :
SCIENCES ET GENIE DES MATERIAUX	K. Wolski Directeur de recherche
MECANIQUE ET INGENIERIE	S. Drapier, professeur
GENIE DES PROCEDES	F. Gruy, Maître de recherche
SCIENCES DE LA TERRE	B. Guy, Directeur de recherche
SCIENCES ET GENIE DE L'ENVIRONNEMENT	D. Graillot, Directeur de recherche
MATHEMATIQUES APPLIQUEES	O. Roustant, Maître-assistant
INFORMATIQUE	O. Boissier, Professeur
IMAGE, VISION, SIGNAL	JC. Pinoli, Professeur
GENIE INDUSTRIEL	A. Dolgui, Professeur
MICROELECTRONIQUE	S. Dauzere Peres, Professeur

EMSE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

ABSI	Nabil	CR		CMP
AVRIL	Stéphane	PR2	Mécanique et ingénierie	CIS
BATTON-HUBERT	Mireille	PR2	Sciences et génie de l'environnement	FAYOL
BERGER DOUCE	Sandrine	PR2		FAYOL
BERNACHE-ASSOLLANT	Didier	PR0	Génie des Procédés	CIS
BIGOT	Jean Pierre	MR(DR2)	Génie des Procédés	SPIN
BILAL	Essaid	DR	Sciences de la Terre	SPIN
BOISSIER	Olivier	PR1	Informatique	FAYOL
BORBELY	Andras	MR(DR2)	Sciences et génie des matériaux	SMS
BOUCHER	Xavier	PR2	Génie Industriel	FAYOL
BRODHAG	Christian	DR	Sciences et génie de l'environnement	FAYOL
BURLAT	Patrick	PR2	Génie Industriel	FAYOL
COURNIL	Michel	PR0	Génie des Procédés	DIR
DARRIEULAT	Michel	IGM	Sciences et génie des matériaux	SMS
DAUZERE-PERES	Stéphane	PR1	Génie Industriel	CMP
DEBAYLE	Johan	CR	Image Vision Signal	CIS
DELAFOSSE	David	PR1	Sciences et génie des matériaux	SMS
DESRAYAUD	Christophe	PR2	Mécanique et ingénierie	SMS
DOLGUI	Alexandre	PR0	Génie Industriel	FAYOL
DRAPIER	Sylvain	PR1	Mécanique et ingénierie	SMS
FEILLET	Dominique	PR2	Génie Industriel	CMP
FRACZKIEWICZ	Anna	DR	Sciences et génie des matériaux	SMS
GARCIA	Daniel	MR(DR2)	Génie des Procédés	SPIN
GERINGER	Jean	MA(MDC)	Sciences et génie des matériaux	CIS
GOEURIOT	Dominique	DR	Sciences et génie des matériaux	SMS
GRAILLOT	Didier	DR	Sciences et génie de l'environnement	SPIN
GROSSEAU	Philippe	DR	Génie des Procédés	SPIN
GRUY	Frédéric	PR1	Génie des Procédés	SPIN
GUY	Bernard	DR	Sciences de la Terre	SPIN
HAN	Woo-Suck	CR	Mécanique et ingénierie	SMS
HERRI	Jean Michel	PR1	Génie des Procédés	SPIN
KERMOUCHE	Guillaume	PR2	Mécanique et Ingénierie	SMS
KLOCKER	Helmut	DR	Sciences et génie des matériaux	SMS
LAFOREST	Valérie	MR(DR2)	Sciences et génie de l'environnement	FAYOL
LERICHE	Rodolphe	CR	Mécanique et ingénierie	FAYOL
LI	Jean-Michel			CMP
MALLIARAS	Georges	PR1	Microélectronique	CMP
MOLIMARD	Jérôme	PR2	Mécanique et ingénierie	CIS
MONTHEILLET	Frank	DR		SMS
MOUTTE	Jacques	CR		SPIN
NIKOLOVSKI	Jean-Pierre			CMP
PIJOLAT	Christophe	PR0	Génie des Procédés	SPIN
PIJOLAT	Michèle	PR1	Génie des Procédés	SPIN
PINOLI	Jean Charles	PR0	Image Vision Signal	CIS
POURCHEZ	Jérémy	CR	Génie des Procédés	CIS
ROBISSON	Bruno			CMP
ROUSSY	Agnès	MA(MDC)		CMP
ROUSTANT	Olivier	MA(MDC)		FAYOL
ROUX	Christian	PR		CIS
STOLARZ	Jacques	CR	Sciences et génie des matériaux	SMS
TRIA	Assia		Microélectronique	CMP
VALDIVIESO	François	MA(MDC)	Sciences et génie des matériaux	SMS
VIRICELLE	Jean Paul	MR(DR2)	Génie des Procédés	SPIN
WOLSKI	Krzystof	DR	Sciences et génie des matériaux	SMS
XIE	Xiaolan	PR1	Génie industriel	CIS
YUGMA	Gallian	CR		CMP

ENISE : Enseignants-chercheurs et chercheurs autorisés à diriger des thèses de doctorat (titulaires d'un doctorat d'État ou d'une HDR)

BERGHEAU	Jean-Michel	PU	Mécanique et Ingénierie	ENISE
BERTRAND	Philippe	MCF	Génie des procédés	ENISE
DUBUJET	Philippe	PU	Mécanique et Ingénierie	ENISE
FEULVARCH	Eric	MCF	Mécanique et Ingénierie	ENISE
FORTUNIER	Roland	PR	Sciences et Génie des matériaux	ENISE
GUSSAROV	Andrey	Enseignant contractuel	Génie des procédés	ENISE
HAMDI	Hédi	MCF	Mécanique et Ingénierie	ENISE
LYONNET	Patrick	PU	Mécanique et Ingénierie	ENISE
RECH	Joël	PU	Mécanique et Ingénierie	ENISE
SMUROV	Igor	PU	Mécanique et Ingénierie	ENISE
TOSCANO	Rosario	PU	Mécanique et Ingénierie	ENISE
ZAHOUANI	Hassan	PU	Mécanique et Ingénierie	ENISE

Mise à jour : 03/12/2013

PR 0	Professeur classe exceptionnelle
PR 1	Professeur 1ère classe
PR 2	Professeur 2ème classe
PU	Professeur des Universités
MA (MDC)	Maître assistant
DR	Directeur de recherche

Ing.	Ingénieur
MCF	Maître de conférences
MR (DR2)	Maître de recherche
CR	Chargé de recherche
EC	Enseignant-chercheur
IGM	Ingénieur général des mines

SMS	Sciences des Matériaux et des Structures
SPIN	Sciences des Processus Industriels et Naturels
FAYOL	Institut Henri Fayol
CMP	Centre de Microélectronique de Provence
CIS	Centre Ingénierie et Santé

Table des matières

Liste des tableaux

Table des figures

Introduction générale

L'industrie micro-électronique est devenue dans les dernières décennies l'une des plus importantes du monde. Elle consiste en la fabrication et l'encapsulation de circuits intégrés. Le processus de fabrication d'un circuit intégré se décompose principalement en quatre étapes (fig. 1) :

1. La fabrication des circuits intégrés sur des disques de silicium, appelés *plaques* ou *plaquettes* (ou encore *wafers*, selon la dénomination anglo-saxonne).
2. Le test des puces sur les plaquettes.
3. L'assemblage, où sont découpées les plaques en circuits individuels qui seront encapsulés et connectés à l'appareil final.
4. le test du produit final.

Les deux premières étapes constituent la phase de fabrication des plaquettes, appelée *front-end* et les deux étapes suivantes composent la phase d'encapsulation et d'assemblage du produit, qu'on appelle *back-end*.

Les plaquettes de silicium sont les éléments de base pour la fabrication des microprocesseurs (fig. 2). Les progrès apportés dans leur traitement ont notamment contribué à réduire le coût des ordinateurs. La demande toujours croissante de produits électroniques modernes a entraîné la multiplication d'usines spécialisées dans ce domaine. Malgré ce développement important, les problématiques de planification de production et d'ordonnancement ne cessent de se poser (voir Uszoy [84]).

La phase la plus longue (et aussi la plus coûteuse) est le *front-end*, consistant à traîter les plaquettes de silicium. En effet, le coût d'une nouvelle usine de fabrication de *wafers* avoisine les 4 milliards de dollars et il faut entre 4 et 6 semaines pour obtenir une plaquette avec tous ses circuits intégrés [32] (voir figure 2). Une opération aussi critique mérite qu'on s'y attarde et c'est ce à quoi nous nous sommes intéressés dans ces travaux de thèse.

Précisons qu'il s'agit probablement là des processus de production les plus complexes qui existent (voir [63] et [4]). Cette complexité est en grande partie dûe à une diversité de produits élevée, un nombre important d'étapes et d'opérations dans la route (i.e. la gamme opératoire) d'un produit, pouvant aller jusqu'à plusieurs centaines, un nombre élevé de machines, etc.

La fabrication d'un lot de plaquettes se scinde en de nombreuses étapes, qui sont réparties dans différents ateliers de l'usine. L'atelier de *photolithographie*, l'une des

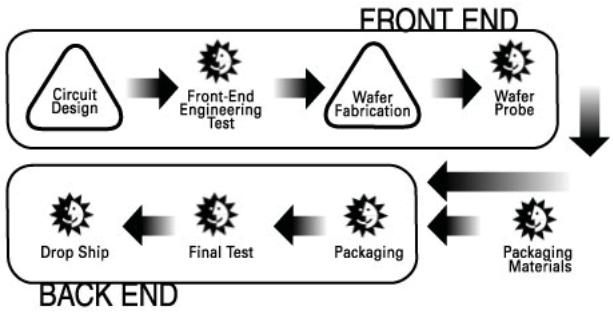

FIGURE 1 – Les différentes étapes de production. Le traitement des wafers est opéré dans le *front-end* et la phase d'assemblage du produit final est désignée par le *back-end*.

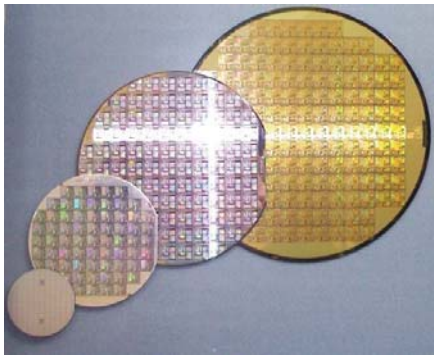

FIGURE 2 – Plaquettes de silicium, aussi appelées *wafers*, avec circuits intégrés

étapes de fabrication, est celui qui nous intéresse dans ce travail et nous verrons pourquoi au chapitre 1. L'objectif a été de modéliser et résoudre un problème d'ordonnancement issu de l'atelier de photolithographie. Le chapitre 1 présente une formalisation du problème. Les choix effectués sont motivés et un aperçu de l'état actuel des travaux dans ce domaine est exposé. Le chapitre 2 détaille l'analyse de la complexité des problèmes abordés. Des méthodes de résolution exacte sont présentées au chapitre 3 et des algorithmes approchés au chapitre 4. Une approche plus appliquée est abordée au chapitre 5 avec l'étude multicritère du problème, qui utilise les résultats obtenus pour la résolution du problème monocritère. La mise en œuvre de ce travail à travers un logiciel d'optimisation et d'aide à la décision pour une entreprise fera l'objet du chapitre 6.

Notations

- n : nombre de tâches
- J : ensemble des tâches
- m : nombre de machines
- M : ensemble des machines
- ℓ : nombre de ressources auxiliaires (masques)
- A : ensemble de ressources auxiliaires
- c_j : nombre de plaquettes de la tâche j
- w_j : priorité de la tâche j
- φ_j : masque, ressource auxiliaire requise par la tâche j
- $\mathcal{M}_j$: ensemble de machines *qualifiées* pour la tâche j
- ρ_{ij} : durée d'exécution de la tâche i par la machine j
- β_{ijk} : temps de setup mis par la machine j pour exécuter la tâche k juste après la tâche i
- H : horizon de temps
- R_i : emplacement initial de la ressource auxiliaire i
- n_i : nombre de tâches nécessitant la ressource auxiliaire i
- E_i : ensemble des tâches nécessitant la ressource auxiliaire i
- Q_j : ensemble des tâches pour lesquelles la machine j est qualifiée.
- $M_j(i)$: ensemble des tâches de E_i pour lesquelles la machine j est qualifiée.

Chapitre 1

Contexte industriel et scientifique

Dans ce chapitre, le problème d'ordonnancement issu de l'atelier de photolithographie en fabrication de semi-conducteurs est défini. C'est ce problème qui est traité tout au long de la thèse. Les particularités de la zone de photolithographie sont détaillées afin de justifier la modélisation qui en a été faite. De même, les fonctions objectif les plus pertinentes au vu du contexte sont décrites et formalisées. Un autre objectif de ce chapitre est de dérouler un état de l'art sur les travaux qui ont été menés dans ce sens, afin de situer nos travaux en termes d'originalité et de complexité.

1.1 Contexte industriel

La fabrication de semi-conducteurs est probablement le processus industriel le plus complexe qui existe. La multiplicité et la diversité des étapes de fabrication des produits rendent impossible l'ordonnancement global de l'unité de fabrication. En effet, les contraintes régissant le fonctionnement des machines ou la gestion de ressources supplémentaires changent d'un atelier à l'autre de l'unité de fabrication. La fabrication d'un composant à semi-conducteurs comprend donc de nombreuses phases très techniques et spécialisées. Les opérations de base (oxydation, déposition, diffusion, photolithographie, gravure, etc. voir figure 1.1) doivent être répétées de nombreuses fois (plus de 40 fois pour les produits les plus avancés) avant que le circuit ne soit terminé. La réalisation des microcircuits part d'une plaquette de silicium de très haute pureté. À l'état parfaitement pur, le silicium est presque un isolant, mais certaines impuretés, appelées dopants, ajoutées en quantités variant de 10 à 100 parties par million (ppm), le rendent électriquement conducteur.

Un circuit intégré est constitué de millions de transistors faits de silicium dopé. Ils sont tous reliés selon un type de circuit spécifique. Des centaines de microcircuits peuvent être réalisés sur une même plaquette. Parmi les grandes étapes de fabrication de ces composants, on compte l'oxydation, la photolithographie, la gravure chimique, le dopage (implant), la diffusion, le dépôt chimique ou encore le nettoyage. Les étapes de tests, de marquage, d'emballage et d'expédition sont postérieures aux phases de fabrication et sont en général effectuées dans d'autres usines.

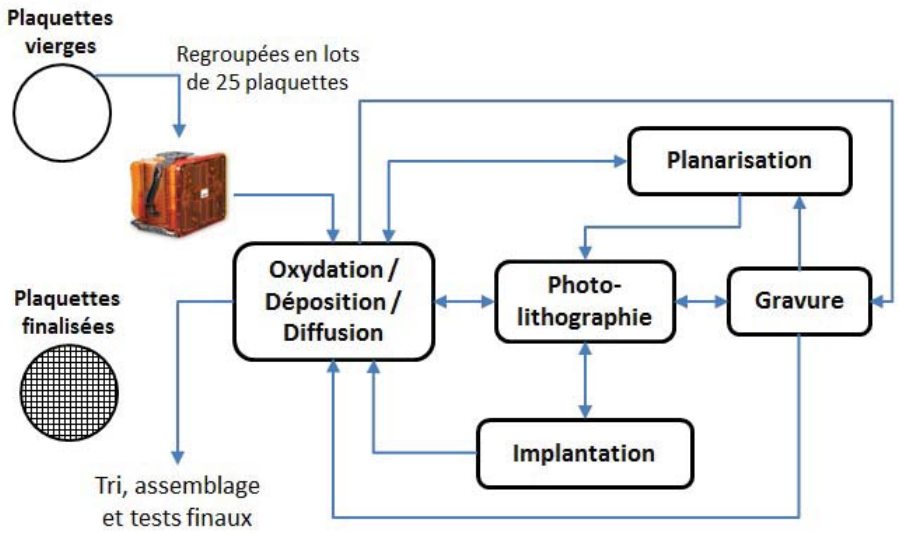

FIGURE 1.1 – Ré-entrance des flux dans les ateliers de fabrication en semi-conducteur.

Le *dépôt* consiste en l'application d'une couche mince sur la surface de la plaquette, qui est retirée à certains endroits pendant la *gravure*. Une *résine photosensible* est posée, en prévision de l'étape de *photolithographie*, au cours de laquelle un motif spécifique, celui du circuit, est greffé sur la couche photosensible. Cette étape est critique et nous verrons pourquoi par la suite. L'étape de *nettoyage* permet notamment de retirer les couches ou particules non désirées de la plaque.

Au delà des nombreuses étapes, plusieurs éléments rendent la planification de la production difficile à mettre en œuvre. D'abord, les plaquettes, qui sont regroupées en *lots* de 25 dans des conteneurs adaptés, doivent visiter un même atelier de nombreuses fois. On dit que les flux sont ré-entrants dans la route d'un lot (figure 1.1). Par ailleurs, les types d'équipements sont très variés : par exemple, le fonctionnement en batch (plusieurs lots peuvent être traités simultanément sur une machine) s'oppose aux processus qui traîtent une plaquette à la fois (*single wafer process*). Les machines fonctionnent en parallèle et sont différentes en termes de vitesse de traitement. Elles sont souvent sujettes à des maintenances préventives, et peuvent être imprévisibles. Ces machines sont par ailleurs extrêmement chères. Elles constituent donc une ressource limitée.

En photolithographie, de nombreuses décisions doivent être prises et cet atelier est très souvent l'atelier limitant de la zone. Les machines ont un coût très élevé

en fabrication de semi-conducteurs. La photolithographie abrite les équipements les plus coûteux de l'unité de production. Leur prix peut culminer à 40 millions de dollars. En particulier, les *steppers* sont des machines extrêmement onéreuses et le processus est donc *bottleneck*. Pour cette raison, la photolithographie est bien la partie la plus critique [1].

Cette étape se compose du dépôt de résine, de l'exposition lumineuse et du développement. Un polymère photosensible est appliqué sur les plaquettes et des motifs en trois dimensions sont tracés sur leur surface. Cela est fait au moyen d'un *masque*, ou *réticule*, qui comme un pochoir, détermine avec précision le motif à graver (fig. 1.2). Ce procédé s'exécute sur les *steppers* qui se chargent de l'exposition à un rayonnement ultra-violet. Le masque détermine donc les parties du *wafer* qui seront exposées et ainsi polymérisées via la résine. Ces parties seront ensuite retirées pendant la phase de développement. Les circuits intégrés étant composés de couches, ce procédé peut se répéter jusqu'à 40 fois. Un nombre très important de contraintes doit être considéré dans un tel atelier. Les machines ne sont pas toujours qualifiées pour tous les types de traitements et deux exécutions consécutives sont parfois séparées par des périodes de changement de température (qu'on nomme *setup* dans la littérature dédiée à l'ordonnancement).

Les masques sont très chers et ne sont par conséquent pas disponibles en grand nombre. Ce sont donc des ressources partagées. Le nombre important de masques qu'il faut gérer (plusieurs milliers) aussi bien pour le déplacement que pour le rangement, constitue une réelle problématique, très liée à l'ordonnancement dans la zone. En effet, même dans des unités de fabrication à transport automatisé, les masques sont transportés de façon manuelle par des opérateurs. Il est alors nécessaire d'apporter des solutions de planification qui intègrent une bonne gestion des masques, qui sont des ressources, puisqu'ils interviennent dans le traitement des lots, et des ressources critiques, du fait de leur prix. Les travaux de la présente thèse se sont concentrés sur ces aspects d'ordonnancement incluant, comme on le verra à la section 1.2.3, le souci de produire vite et celui de bien gérer les ressources.

Dans la partie qui suit, les composantes du problème sont détaillées et justifient la modélisation retenue du problème d'ordonnancement qui est étudié.

1.2 Définition du problème

Dans cette section, des notations sont introduites et utilisées tout au long du document pour représenter le problème d'ordonnancement dans l'atelier de photolithographie, appelé plus simplement *problème d'ordonnancement en photolithographie*. Pour chaque contrainte, la correspondance avec la difficulté réelle rencontrée, est précisée. Les critères retenus pour l'optimisation sont justifiés. Le but de cette partie est d'expliquer l'origine du problème étudié. L'apparente complexité n'est

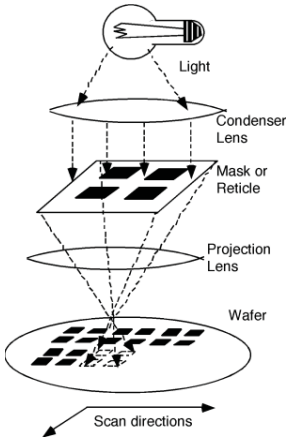

FIGURE 1.2 – La photolithographie, utilisation des réticules

pas due au hasard et traduit bien la difficulté à laquelle on fait face pour ordonnancer correctement les tâches dans cet atelier. Certaines contraintes additionnelles rencontrées dans la réalité sont occultées pour la modélisation, mais seront reprises au chapitre 6 et considérées dans la méthode de résolution adoptée. Cette dernière dérive directement des travaux sur le problème décrit dans cette section.

1.2.1 Notations

Comme dans tout problème d'ordonnancement, on considère un ensemble J de n tâches et un ensemble M de m machines. Les tâches ici modélisent les lots à traiter. Il faut aussi prendre en compte un ensemble A de ℓ ressources auxiliaires (appelées aussi *masques*), nécessaires pour le traitement des tâches sur les machines. Chaque lot possède un nombre spécifique de plaquettes. Précédemment, nous avions avancé qu'il y en avait 25 (nombre maximal dans le conteneur transportant les plaquettes) mais certains lots en contiennent moins. Il faut donc conserver cette donnée. Les lots ont chacun une ressource auxiliaire associée pour effectuer l'opération de photolithographie au sein d'une machine. De plus, il ne faut pas omettre que certaines machines sont qualifiées pour un lot alors que d'autres non. Les machines sont différentes. Elles n'ont pas toutes la même vitesse, la même *configuration*, ne traitent pas toutes le même ensemble de lots, et sont par conséquent à modéliser dans ce sens. C'est pourquoi le problème traité est un problème à machines parallèles différentes. Les temps de changement de température (setup) sont aussi fréquents selon la configuration de la machine et l'enchaînement des lots. On appelle *recette* le schéma de configuration qui lie une machine à un groupe de lots. En d'autres termes, deux lots ayant la même recette s'enchaînent avec un temps de setup négligeable tandis que dans d'autre cas ces temps sont

plus importants.

Enfin, évoquons une notion importante ici, qui est liée au fait que les lots n'ont pas tous le même degré d'urgence. Certains, selon leur temps d'attente, le produit final auquel ils sont affectés, ou encore le client, sont prioritaires. Ce point conduit inévitablement à prévoir dans la gestion de l'atelier des objectifs qui tiennent compte de cette hiérarchie.

Ces éléments ont conduit à utiliser les notations suivantes pour le problème :

- Pour toute tâche j, son nombre de plaquettes $c_j \in \mathbb{N}$, sa priorité w_j, la ressource auxiliaire (masque) $\varphi_j \in A$ requise par cette tâche et l'ensemble de machines *qualifiées*, c'est-à-dire l'ensemble des machines qui peuvent exécuter j, $\mathcal{M}_j \subset M$.
- Pour toute tâche i et toute machine j telle que $j \in \mathcal{M}_i$, le temps d'exécution $\rho_{ij} \in \mathbb{N}$ de i par j.
- Pour tout couple de tâches (i, k) et toute machine $j \in \mathcal{M}_i \cap \mathcal{M}_k$, le temps de setup $\beta_{ijk} \in \mathbb{N}$ mis par la machine j pour exécuter k juste après i.
- Un horizon de temps H.
- L'emplacement initial $R_i \in \{0, \ldots, m\}$ de chaque ressource auxiliaire i, indice de la machine sur laquelle se trouve initialement la ressource A_i, 0 représentant la zone de stockage.

Ces notations couvrent l'ensemble du problème et seront utilisées dans tout le document.

1.2.2 Contraintes

Pour fixer les idées et s'accorder aux définitions habituelles en théorie de l'ordonnancement, les premières contraintes précisent le domaine de résolution.

Non préemption Une tâche j est exécutée une et une seule fois sur une machine qualifiée. Il s'agit d'une contrainte classique d'ordonnancement de tâches sur machines parallèles. Il faut affecter chaque tâche à une ressource unique. Il n'y a pas de préemption, une tâche ne peut pas être interrompue. Il convient donc de poser cette contrainte, car les lots qui passent dans une machine sont destinés à être intégralement traîtés, sans interruption.

Machines disponibles Les dates de début sont entières et toutes les machines sont disponibles à la date 0.

Capacité des machines Une machine ne peut traiter qu'une tâche à la fois. Les machines de photolithographie disposent de ports de chargement multiples et peuvent donc effectuer des traitements en parallèle. Mais l'enchaînement des opérations suit un ordre précis une fois l'étape de chargement des plaquettes passée. C'est pourquoi on peut considérer ces opérations comme séquentielles.

Les contraintes suivantes découlent des spécificités de l'atelier de photolithographie et donnent au problème son caractère original et complexe.

Contraintes de setup Notons qu'il y a des contraintes de setup dépendant de la séquence des tâches et de la machine. Les réglages des machines, les conditions de température, la compatibilité avec les lots (recettes), sont autant de facteurs qui entraînent l'apparition de temps de setup dans l'enchaînement des lots.

Ressources auxiliaires De plus, deux tâches dont les masques sont identiques ne peuvent pas être réalisées simultanément, même sur deux machines différentes. En effet, le coût des masques rendant cette ressource rare dans l'atelier, on est amené à considérer qu'il n'existe qu'un exemplaire d'un type donné. Ici, on considère donc les masques comme des ressources auxiliaires. Dans notre problème, deux types de contraintes sont liées à des ressources : les contraintes liées aux capacités des machines, et celles liées aux ressources auxiliaires. On ne peut associer la même ressource à deux tâches simultanément.

Transfert de ressources Enfin, le déplacement d'un masque d'une machine à une autre consomme une durée unitaire de déplacement. Ces transferts sont assurés dans la plupart des cas par des opérateurs. Ces temps de déplacement sont difficiles à modéliser et sont en moyenne relativement réduits. Le choix a été fait de considérer ces temps unitaires entre deux emplacements différents.

Certaines simplifications sont faites par rapport à la réalité. Des contraintes supplémentaires doivent être considérées pour la mise en œuvre industrielle. Ces contraintes ne seront reprises qu'au chapitre 6.

Stock de masques Les machines ont une capacité de stockage de masques limitée. En effet, si un masque est requis pour l'exécution d'une tâche, il faut vérifier s'il lui reste de la place.

Conteneurs de masques Les masques sont stockés dans des conteneurs communs et chaque conteneur peut en renfermer plusieurs (5 au plus dans notre cas d'étude). Cela implique qu'il peut exister des contraintes d'utilisation non simultanée des masques. En d'autres termes, dans notre cas, utiliser un masque en bloque quatre autres.

Disponibilité des tâches Les tâches ne sont pas toutes disponibles à la même date. Dans un contexte de planification en temps réel, il faut être capable de tenir compte de cette information, pour modifier l'ensemble des tâches à ordonnancer.

Enchaînement des tâches Les machines disposent de ports de chargement multiples (jusqu'à 4 en général) qui permettent de gérer simultanément plusieurs

lots à traiter. Ainsi les dates de début suivent une règle différente du cas de machines à capacité unique. Ce point sera détaillé au chapitre 6.

Temps de transport Jugés négligeables dans la modélisation, on les suppose constants pour les ressources auxiliaires et nuls pour les tâches. Dans la réalité, les tâches sont des lots, qu'il faut physiquement transporter à leur machine affectée, de même pour les masques. Ainsi, on aura une matrice de temps de transport.

Maintenances Les machines ont des périodes de maintenance. Dans ces intervalles, aucune tâche ne peut se terminer ni commencer. Ces périodes peuvent aussi correspondre à des imprévus, comme dans le cas de pannes. Ces données sont prises en compte dans la mise en œuvre industrielle.

Cette liste de contraintes additionnelles est occultée dans les chapitres 2 à 5.

1.2.3 Critères

La singularité du problème réside en grande partie dans les critères étudiés. Ceux-ci couvrent des objectifs intrinsèquement adaptés au fonctionnement de l'atelier de photolithographie. Ils concernent la production de plaquettes, les priorités des produits et les transferts de ressources auxiliaires entre machines. Nous avons défini trois principaux objectifs à optimiser en pratique.

Minimiser la somme pondérée des dates de fin Il faut considérer des priorités entre les tâches à cause de leur temps d'attente dans l'unité de production et parce qu'elles sont liées à des produits et des clients différents. Par conséquent, il est nécessaire de tenir compte d'un critère qui intègre cette hiérarchie entre les lots à traiter. C'est pourquoi la somme pondérée des dates de fin est retenue. Si on note C_j la date de fin d'une tâche j, la fonction objectif est $\sum_{j=1}^{n} w_j C_j$. C'est un critère classique en théorie de l'ordonnancement. Les travaux liés seront évoqués en section 1.3.

Maximiser le nombre de plaquettes avant un horizon S'il n'y a pas de limite sur l'horizon de planification, ce critère n'a pas de sens, puisque toutes les tâches sont vouées à être traitées par une machine. Comme les tâches arrivent de façon continue dans l'atelier, il est pertinent de spécifier un horizon de temps limité H, afin de fixer un ordonnancement sur une tranche temporelle, sans qu'il soit remis en cause par des lots arrivant plus tard. De plus, les lots ordonnancés après l'horizon H pourraient impacter négativement l'ordonnancement alors qu'ils pourraient être placés avec des lots arrivant plus tard. C'est pourquoi il est pertinent de maximiser le nombre de plaquettes produites dans la limite imposée par l'horizon H. Si, dans une solution, on note m_j la machine sur laquelle est affectée la tâche j et t_j sa

date de début, alors le nombre de plaquettes produites avant H s'exprime ainsi :

$$\sum_{j=1}^{n} c_j \theta_j$$

où θ_j est le taux d'accomplissement de la tâche j avant H.

$$\theta_j = \begin{cases} \frac{\min(\rho_{jm_j}, H - t_j)}{\rho_{jm_j}} & \text{si } t_j \leq H, \\ 0 & \text{si } t_j > H. \end{cases}$$

Cette expression modélise le fait que le nombre de plaquettes produites est proportionnel au temps de traitement.

Minimiser le nombre de transferts de ressources auxiliaires Dans la zone de fabrication, transférer un masque nécessite un opérateur, qui doit alors interrompre son travail. Une solution qui minimise le nombre d'interventions de ce type est d'un grand intérêt pour la fluidité du processus et diminue certains risques (accidents, chute des masques). Ce critère est rendu pertinent et intéressant à analyser par les contraintes de qualification de machines. En effet, si les machines pouvaient exécuter toutes les tâches, une solution évidente qui n'occasionne aucun déplacement de ressource auxiliaire pourrait être construite.

1.3 État de l'art

On étudie un problème d'ordonnancement avec machines parallèles différentes, des contraintes d'*éligibilité*, des ressources auxiliaires partagées et des temps de setup dépendant de la séquence et de la machine. Trois fonctions objectif sont retenues : la somme pondérée des dates de fin, le nombre de plaquettes traitées dans un horizon de temps fixé et le nombre de transferts de ressources auxiliaires. Les deux derniers cités n'ont à notre connaissance jamais été abordés dans la littérature sur l'ordonnancement et une contribution de notre travail est d'analyser ces critères. Comme on le verra, peu de sources traitent de problèmes combinant temps de setup dépendant de la séquence et ressources auxiliaires, encore moins avec des fonctions objectif aussi peu conventionnelles. Dans ce qui suit, un survol général des travaux en ordonnancement traitant de problèmes à machines parallèles est effectué, les principaux résultats étant exposés. Puis des recherches sur des cas se rapprochant du problème défini précédemment sont évoquées afin de situer l'étude faite dans cette thèse.

1.3.1 Problèmes avec machines parallèles

Avant d'aborder le vrai sujet de cette section, à savoir l'étude des problèmes d'ordonnancement à machines parallèles, on se concentre d'abord sur le cas d'une machine unique. Le problème consistant à optimiser la somme pondérée des dates de fin sur une machine, noté $1||\sum_j w_j C_j$ est résolu simplement par un algorithme glouton consistant à trier les tâches dans l'ordre non-décroissant des ratios $\frac{p_j}{w_j}$, où w_j et p_j sont respectivement le poids et le temps d'exécution de la tâche j. Cette méthode, appelée règle de Smith [79], garantit l'optimalité de la solution obtenue. Le fait de rajouter des dates de disponibilité r_j rend le problème NP-difficile au sens fort, même dans le cas non pondéré [50]. Avec des contraintes de précédence entre les tâches, Lawler [45] et Lenstra [49] montrent que $1|prec|\sum_j w_j C_j$ est NP-difficile [6].

Cette section est donc dédiée aux problèmes à machines parallèles. Elle sera divisée en deux parties selon le type des machines considérées. D'abord on présente un état de l'art des travaux autour du cas de machines identiques (le temps d'exécution d'une tâche est le même sur toutes les machines) puis des résultats plus généraux sur les machines différentes (le temps d'exécution d'une tâche dépend aussi de la machine) sont abordés. Pour chaque partie, on se concentre sur deux critères classiques que sont le *makespan* (ou $C_{\max}$), lié à la durée totale de l'ordonnancement et la somme pondérée des dates de fin.

Le problème $P||C_{\max}$ consiste à placer des tâches sur des machines parallèles identiques de sorte à minimiser la date de fin de l'ordonnancement. Malgré son apparente simplicité, ce problème est NP-difficile. En effet, on peut montrer que le cas particulier à deux machines est, dans sa version de décision, au moins aussi difficile que le problème de partition [42]. Ce résultat donne cours à des idées d'algorithmes d'approximation basés sur des listes de priorités [29]. On trie toutes les tâches selon un critère prédéfini et on les place dans cet ordre sur la machine disponible au plus tôt. L'algorithme LPT consistant à trier les tâches dans l'ordre décroissant de leur temps d'exécution a été étudié par Graham [30]. Il possède un rapport d'approximation de $\frac{4}{3} - \frac{1}{3m}$, où m est le nombre de machines. Graham exhibe un cas limite où ce rapport est atteint. On voit ici que plus le nombre de machines augmente, plus on se rapproche du pire ratio d'approximation possible, qui est $\frac{4}{3}$. Dans le pire des cas, l'algorithme LPT peut fournir une solution 33% plus longue que l'optimum. Le problème est NP-difficile, mais pas au sens fort, comme le montre Rothkopf [74], qui présente un algorithme de programmation dynamique pseudo-polynomial pour résoudre $P||C_{\max}$ en temps $O(nC^m)$, où n est le nombre de tâches, m est le nombre de machines et C est une borne supérieure du makespan optimal.

Toujours dans le cas de machines parallèles, le critère $\sum_j C_j$ de la somme des dates de fin des tâches est plus simple à appréhender. Le problème $P||\sum_j C_j$ est

polynomial et se résout en $O(n \log n)$ par l'algorithme *SPT* consistant à placer les tâches dans l'ordre croissant de leur durée d'exécution [16]. Il suffit d'introduire des poids associés aux tâches pour que, même dans le cas de deux processeurs, le problème, noté $P2||\sum_j w_j C_j$ devienne NP-difficile [10]. Kawaguchi et Kyan [43] montrent que l'algorithme *WSPT*, consistant à trier les tâches dans l'ordre décroissant du ratio $\frac{w_j}{p_j}$ fournit un rapport d'approximation $\frac{1}{2}(1+\sqrt{2}) \approx 1,207$, ce qui constitue le meilleur rapport connu pour le problème $P||\sum_j w_j C_j$. Woeginger et Skutella [91] proposent à leur tour le premier schéma d'approximation polynomial (PTAS) pour ce problème. Sahni [75] donne un *FPTAS* (*schéma d'approximation totalement polynomial*) pour la version du problème où le nombre de machines m est fixé. Tout comme pour le critère du makespan, on peut relever quelques approches pseudo-polynomiales. Lee et Uszoy [47] donnent un algorithme de programmation dynamique pseudo-polynomial. La complexité de leur méthode dépend de la somme des poids des tâches, contrairement aux approches classiques, dépendant de leur durée d'exécution. Citons pour finir Lawler et Moore [46] qui fournissent un algorithme de programmation dynamique pseudo-polynomial si le nombre de machines est fixé.

Dans ce qui suit, les machines sont différentes. Ce point rend donc les problèmes étudiés bien plus généraux et complexes à traiter. Il suit naturellement des résultats donnés sur $P||C_{\max}$ que $R||C_{\max}$ est NP-difficile. Il convient de s'intéresser aux algorithmes les plus efficaces pour résoudre ce problème. Lenstra, Shmoys et Tardos [51] construisent une méthode d'approximation basée sur l'arrondi d'une solution obtenue par programmation linéaire. L'algorithme est de rapport 2. Les auteurs montrent que lorsque les durées d'exécution sont dans l'ensemble $\{1, 2\}$, le problème d'ordonnancement avec machines parallèles différentes pour minimiser le makespan est polynomial. Dans tout autre cas que celui où les temps d'exécution sont dans des ensembles de la forme $\{p, 2p\}$, où p est un entier quelconque, le problème est NP-difficile. Ainsi, tous les cas particuliers polynomiaux du problème sont exhibés. Ibarra et Kim [38] étudient le cas à deux processeurs pour lequel un algorithme de complexité $O(n \log n)$ est donné avec un rapport d'approximation de $2(1+\sqrt{5})$.

Le critère $\sum_j C_j$ laisse plus d'espoir. Bruno, Coffman et Sethi [10] déroulent un algorithme en $O(n^3)$ pour résoudre le problème $R||\sum C_j$, montrant ainsi qu'il est polynomial. Leur méthode est basée sur la résolution d'un problème de transport avec capacités sur les arcs. Les résultats présentés précédemment montrent que $R||\sum w_j C_j$ est NP-difficile. Peu de travaux ont abouti sur des résultats concernant des rapports d'approximation encourageants.

1.3.2 Problèmes avec temps de setup

Les problèmes d'ordonnancement incluent parfois, comme on l'a vu en préambule du document, des contraintes associées à l'enchaînement des tâches sur les ressources. On les retrouve très souvent dans la réalité, et les problèmes d'ordonnancement dans lesquels ils ne sont pas considérés découlent du fait qu'ils sont négligeables en comparaison aux temps de traitement des tâches ou inclus dans ces durées. Ces contraintes peuvent être, comme dans le cas de la photolithographie, inhérentes au fonctionnement et à la compatibilité entre les machines et les tâches à exécuter. De nombreux travaux ont été menés en ordonnancement avec temps de setup. Allahverdi et al. [3] présentent un état de l'art récent des recherches sur ce type de problème. Tous types d'ateliers sont considérés (job shop, open shop, flow shop, etc) ainsi que les cas de setup dépendant ou non de la séquence des tâches. Deux degrés de séparation sont considérés, les problèmes avec ou sans formation de batch, et les setups dépendant ou non de la séquence des tâches, ce qui constitue une classification de quatre grandes familles. Dans notre problème, les lots ne sont pas regroupés en sous ensembles de la même famille pour être exécutés en parallèle (comme c'est le cas pour le *p-batch*) et des setups dépendant de la séquence des tâches sont considérés. De nombreuses fonctions objectif sont traitées dans ce large spectre de travaux, dont la somme pondérée (et non pondérée) des dates de fin d'exécution. Koulamas et al. [44] traitent le cas de setup dépendant de la séquence ainsi que des tâches déjà exécutées, sur une machine, et montrent qu'on minimise le makespan et la somme des dates de fin en temps $O(n \log n)$ au moyen d'un simple tri. Ce problème, bien que traitant de setup dépendant de la séquence et minimisant une fonction objectif qui nous intéresse, est à une machine est très simple en comparaison. Concernant les problèmes à machines parallèles, Gendreau et al. [27], et Mendes et al. [58] traitent le cas de machines identiques, avec setups dépendant de la séquence pour minimiser le makespan. Les premiers proposent des bornes inférieures et une heuristique, les seconds implémentent deux métaheuristiques, un algorithme de recherche taboue et un algorithme mémétique. Webster et Azizoglu [86] considèrent le problème avec des familles de tâches entre lesquelles on doit observer des périodes de setup et proposent deux algorithmes de programmation dynamique pour minimiser la somme pondérée des dates de fin. Ici, les machines sont identiques. Lorsque le nombre de machines et le nombre de familles sont fixés, l'algorithme est pseudo-polynomial. Lorsqu'il s'agit de données du problème, celui-ci est NP-difficile au sens fort. Lorsqu'il n'y a qu'une famille et que le nombre de machines est fixé, le problème est NP-difficile. Donc avec un nombre de familles et de machines fixé, le problème est toujours NP-difficile. Cependant, l'algorithme de programmation dynamique montre que lorsque ces nombres sont fixés et les poids w_j sont unitaires, le problème est dans P puisque cet algorithme devient polynomial en la taille de l'instance. Un résultat très important vient de

Monma et Potts [65] qui étudient le même problème. Ils prouvent que dans le cas à une machine, une séquence optimale vérifie le fait que les tâches de la même famille sont séquencées dans l'ordre croissant des ratios $\frac{p_j}{w_j}$.

Le cas de machines différentes se rapproche plus de la problématique centrale de la thèse mais est évidemment bien plus ardu à traiter. Weng et al. [87] étudient l'impact de sept heuristiques pour minimiser la somme pondérée des dates de fin des tâches, dans la cas de setup dépendant de la séquence, et montrent que l'une d'elles surclasse les autres. Le problème résolu ne contient pas de contraintes de ressources auxiliaires. Obeid et al. [70] utilisent un modèle de programmation linéaire pour résoudre un problème à machines parallèles différentes avec temps de setup dépendant de la famille de la tâche qui précède. Dans ce cas, les temps de setup ne dépendent pas de la séquence. Rabadi et al. [72] proposent une métaheuristique résolvant ce dernier cas, et améliorant les méthodes existantes sur de grandes instances. Cette dernière problématique rejoint le problème étudié ici, mais n'inclut pas les contraintes de ressources auxiliaires ni les contraintes de qualification (éligibilité) des machines. Par ailleurs, précisons que ces travaux ne traitent pas de temps de setup dépendant de la machine, ce qui est le cas dans la présente problématique.

1.3.3 Problèmes avec ressources auxiliaires

Les problèmes d'ordonnancement qui abordent des contraintes de ressources auxiliaires ont été introduits dans la classification générale en vigueur. La nomenclature de ces problèmes est décrite par Blazewicz et al. [7]. Les contraintes de ressources auxiliaires s'écrivent : $res = \lambda\gamma\rho$. Trois données sont à préciser, à savoir :

- Le nombre de ressources différentes (λ).
- La quantité disponible de ressources de chaque type s_h (γ).
- La quantité de ressources d'un type, requise par chaque tâche r_{hj} (ρ).

Lorsque l'information est une donnée du problème, un point (.) est écrit. Attention toutefois, dans notre version du problème, on ne peut pas vraiment écrire $res = .11$. En effet, le nombre ℓ de ressources auxiliaires (masques) est une donnée, chacune des ressources est disponible en unique exemplaire et chaque tâche a besoin d'un exemplaire pour son exécution, mais il y a une restriction supplémentaire qui n'est pas indiquée par $res = .11$, à savoir le fait qu'une tâche ne requiert qu'un type de ressource à la fois. Dans ce cas particulier, la contrainte de ressource se ramène à la non simultanéité d'exécution de deux tâches nécessitant la même ressource. Blazewicz et al. [7] donnent aussi des résultats de complexité de problèmes pour minimiser le makespan avec temps d'exécution unitaires.

- $P3|res.11, p_j = 1|C_{\max}$ et $P3|res.11, p_j = 1|\sum C_j$ sont NP-difficiles au sens

fort.
- $Q2|res.11, p_j = 1|C_{\max}$ et $Q2|res.11, p_j = 1|\sum C_j$ sont NP-difficiles au sens fort.
- $P2|res..., p_j = 1|C_{\max}$ est polynomial et peut être résolu en $O(\ell n^2 + n^{5/2})$.

D'autres variantes ont été étudiées, par exemple par Grigoriev et al. [31]. Le problème étudié est à machines parallèles différentes, avec des ressources auxiliaires en quantité k. Plus la ressource est allouée en quantité sur la machine, plus le temps d'exécution est court. Les auteurs proposent un algorithme d'approximation de rapport $4 + 2\sqrt{2}$ basé sur une méthode d'arrondi en deux phases à partir d'un programme linéaire. L'algorithme proposé peut s'adapter à la variante avec machines dédiées, où l'on observe une amélioration du rapport, qui passe à $3 + 2\sqrt{2}$. La plupart des articles traitant ce type de problèmes proposent des métaheuristiques [81] [35]. Les versions étudiées sont plus ou moins éloignées du problème rencontré en photolithographie.

Avant de clore cette section, soulignons l'importance des contraintes de qualification des machines. Dans la théorie, on les désigne souvent par la notion d'*éligibilité*. Dans ce domaine, quelques travaux [78] ont par exemple relevé un algorithme d'approximation de rapport $2 - \frac{1}{m}$ pour la minimisation du makespan sur machines identiques. Brucker et al.[9] montrent que minimiser la somme des dates de fin sous ces contraintes est un problème polynomial sur machines identiques. Leung et al. [53] proposent un survol très complet des travaux incluant ce type de contraintes.

Notons que la totalité des travaux cités ici ne concerne que le critère du makespan et de la somme (pondérée ou non) des dates de fin des tâches. La maximisation du nombre de plaquettes des lots, qui est ici une spécificité issue de l'application industrielle, est un critère difficile à situer dans les travaux existants. On verra dans la suite la relation qui existe avec le critère du makespan. Maximiser le nombre de plaquettes traitées avant H peut être aussi vu comme minimiser le nombre (pondéré par la priorité) des tâches en retard, H étant la date d'échéance commune à toutes les tâches. Seulement, ici on considère aussi les tâches qui commencent avant et se terminent après H, ce qui ne colle donc pas parfaitement à ce modèle. On peut par exemple citer Dauzère-Pérès et Sevaux [17] ainsi que M'Hallah et Bulfin [60], qui proposent des méthodes exactes pour minimiser le nombre de tâches en retard sur une machine. La minimisation du nombre de transferts de ressources auxiliaires (masques) est, à notre connaissance, un critère nouveau.

1.4 Conclusion

Le problème d'ordonnancement posé par la zone de photolithographie peut s'exprimer, en théorie de l'ordonnancement, de la manière présentée dans ce cha-

pitre. La modélisation qui en a été faite satisfait à la double nécessité d'être en accord avec les problématiques classiques tout en intégrant au mieux la complexité réelle du problème. La combinaison de contraintes difficiles à gérer, mais aussi la prise en compte de critères originaux, nous laisse peu de matière quant à l'état de l'art sur ce sujet. Les travaux entrepris, qui seront présentés dans les chapitres suivants, s'articulent autour de l'étude de complexité de ces problèmes nouveaux et de méthodes de résolution, qui seront analysées.

Chapitre 2

Analyse de complexité

Dans ce chapitre, on montre que le problème d'ordonnancement en photolithographie est NP-difficile au sens fort, pour chacune des trois fonctions objectif que sont le nombre de plaquettes traitées avant une date donnée (horizon), la somme pondérée des dates de fin d'exécution et le nombre de déplacements de ressources auxiliaires. Ces résultats permettent par la suite de justifier les approches de résolution adoptées pour ces trois problèmes.

2.1 Introduction

Le problème défini au chapitre précédent est un problème d'ordonnancement sur machines parallèles différentes. Il comprend diverses contraintes issues d'une zone critique dans les usines de fabrication de semi-conducteurs, l'atelier de photolithographie. C'est un problème avec ensembles de machines éligibles (chaque tâche a un sous-ensemble de machines éligibles pour l'exécuter) et temps de setup dépendant de la séquence des tâches et de la machine. Par ailleurs, chaque tâche possède une ressource auxiliaire attribuée pour son exécution. Par conséquent, des contraintes de ressources auxiliaires sont également présentes : deux tâches nécessitant la même ressource ne peuvent s'exécuter en parallèle. Enfin, le déplacement d'une ressource entre deux machines différentes requiert une durée unitaire de transfert.

Ce problème est traité pour trois critères distincts et complémentaires, couvrant une majorité des objectifs rencontrés quotidiennement dans le contexte industriel :

1. Le nombre de plaquettes produites (chaque tâche a un nombre de plaquettes associé) avant une date donnée (horizon), à maximiser.
2. La somme pondérée des dates de fin ($\sum_j w_j C_j$), à minimiser, prend en compte la priorité des tâches.
3. Le nombre de transferts de ressources auxiliaires entre les machines, à minimiser. Ce critère est non régulier.

On rappelle les données du problème :
- Un ensemble de n tâches, ou *jobs* $J = \{J_1, \ldots, J_n\}$.
- Un ensemble de m machines $M = \{M_1, \ldots, M_m\}$.

- Un ensemble de ℓ ressources auxiliaires $A = \{A_1, \ldots, A_\ell\}$, $\ell \leq n$.
- Pour toute tâche J_i, son nombre de *plaquettes* $c_i \in \mathbb{N}$, sa priorité w_i, son indice de ressource requise $\varphi_i \in \{1, \ldots, \ell\}$ et son ensemble d'indices de machines qualifiées $\mathcal{M}_i \subset \{1, \ldots, m\}$.
- Pour toute tâche J_i et toute machine M_j telle que $j \in \mathcal{M}_i$, le temps d'exécution $\rho_{ij} \in \mathbb{N}$ de J_i par M_j.
- Pour tout couple de tâches (J_i, J_k) et toute machine M_j, $j \in \mathcal{M}_i \cap \mathcal{M}_k$, le temps de setup $\beta_{ijk} \in \mathbb{N}$ de la machine M_j pour exécuter J_k juste après J_i.
- L'emplacement initial $R_i \in \{0, \ldots, m\}$ de chaque ressource auxiliaire A_i, indice de la machine sur laquelle se trouve initialement la ressource A_i, 0 représentant la zone de stockage des ressources auxiliaires.

On ajoute les notations suivantes :

- Pour toute ressource auxiliaire $A_i \in A$, on note n_i le nombre de tâches de J la nécessitant et $E_i = \{J_{i_1}, \ldots, J_{i_{n_i}}\}$ l'ensemble de ces tâches :

$$E_i = \{J_k \in J \mid \varphi_k = i\}.$$

- Pour toute machine $M_j \in M$, l'ensemble Q_j des tâches pour lesquelles M_j est qualifiée.
- Pour tout machine $M_j \in M$, pour toute ressource auxiliaire $A_i \in A$, l'ensemble

$$M_j(i) = \{k \mid J_k \in Q_j \cap E_i\}$$

  des indices des tâches de E_i pour lesquelles M_j est qualifiée.
  On pose de plus $M_0(i) = \emptyset$ pour tout i, $1 \leq i \leq \ell$.

Dans le reste du chapitre, on notera Π_1 le problème consistant à maximiser le nombre de plaquettes avant H, Π_2 le problème d'optimisation consistant à minimiser la somme pondérée des dates de fin d'exécution et Π_3 le problème de minimisation du nombre de transferts de ressources auxiliaires.

Par ailleurs, soient :

- Π_A le problème $R||\sum w_j C_j$.
- Π'_A le même problème avec la contrainte supplémentaire imposant que la première tâche sur chaque machine commence au plus tôt à $t = 1$, et non à 0.
- Π_B le problème $R||C_{\max}$.
- Π'_B le même problème avec la contrainte supplémentaire imposant que la première tâche sur chaque machine commence au plus tôt à $t = 1$, et non à 0.

2.2 Minimisation de la somme pondérée des dates de fin

L'étude de la complexité du problème Π_2 nécessite une remarque préliminaire :

Proposition 1. *Les problèmes Π_A, Π'_A, Π_B et Π'_B vérifient les énoncés suivants :*

1. *On peut déduire de toute solution optimale de Π_A une solution optimale de Π'_A en temps $O(n)$ et réciproquement.*
2. *On peut déduire de toute solution optimale de Π_B une solution optimale de Π'_B en temps $O(n)$ et réciproquement.*

Démonstration. Démontrons d'abord le premier point.

Soit S^* une solution optimale de Π_A. Son coût est $c(S^*) = \sum_{i=1}^{n} w_i C_i^*$. Considérons la solution dans laquelle toutes les tâches commencent leur exécution une unité de temps après leur date de début d'exécution dans S^*. On note cette solution $\tilde{S}$, qui est alors une solution de Π'_A, et qui se déduit de S^* en $O(n)$. Montrons par l'absurde que $\tilde{S}$ est optimale pour Π'_A.

Le coût de $\tilde{S}$ est $c(\tilde{S}) = \sum_{i=1}^{n} w_i \tilde{C}_i = \sum_{i=1}^{n} w_i C_i^* + \sum_{i=1}^{n} w_i$. Supposons qu'il existe une solution $\hat{S}$ de Π'_A telle que $c(\hat{S}) = \sum_{i=1}^{n} w_i \hat{C}_i < c(\tilde{S})$. Considérons la solution $\hat{S}^*$ correspondante dans laquelle toutes les tâches commencent leur exécution une unité de temps *avant* leur date de début d'exécution dans $\hat{S}$. Il s'agit alors d'une solution de Π_A, de coût :

$$c(\hat{S}^*) = c(\hat{S}) - \sum_{i=1}^{n} w_i < \sum_{i=1}^{n} w_i C_i^* + \sum_{i=1}^{n} w_i - \sum_{i=1}^{n} w_i = c(S^*)$$

d'où la contradiction.

La réciproque se démontre de façon analogue.

Démontrons maintenant le second point.

Soit S^* une solution optimale de Π_B. Son coût est $c(S^*) = C^*_{\text{opt}}$. Considérons la solution dans laquelle toutes les tâches commencent leur exécution une unité de temps après leur date de début d'exécution dans S^*. On note cette solution $\tilde{S}$, qui est alors une solution de Π'_B, et qui se déduit de S^* en $O(n)$. Montrons par l'absurde que $\tilde{S}$ est optimale pour Π'_B.

Le coût de $\tilde{S}$ est $c(\tilde{S}) = \tilde{C}_{\text{opt}} = C^*_{\text{opt}} + 1$. Supposons qu'il existe une solution $\hat{S}$ de Π'_B telle que $c(\hat{S}) = \hat{C}_{\text{opt}} < c(\tilde{S})$. Considérons la solution $\hat{S}^*$ correspondante

dans laquelle toutes les tâches commencent leur exécution une unité de temps *avant* leur date de début d'exécution dans $\hat{S}$. Il s'agit alors d'une solution de Π_B, de coût :

$$c(\hat{S}^*) = c(\hat{S}) - 1 < C^*_{\text{opt}} + 1 - 1 = c(S^*)$$

d'où la contradiction.

La réciproque se démontre de façon analogue.

□

La proposition précédente permet d'établir l'équivalence entre les problèmes Π_A et Π'_A d'une part, et entre Π_B et Π'_B d'autre part. La démonstration en a été détaillée, mais on voit bien que l'écart entre les coûts des solutions est constant et que résoudre un problème revient clairement à résoudre l'autre.

On montre dans ce qui suit que le problème Π_2 est une généralisation du problème Π'_A.

On notera dans la démonstration du théorème 1 ci-dessous que la conclusion sur la complexité de Π_2 se base sur le fait que le problème Π_A est une généralisation de $P||\sum w_i C_i$, NP-difficile au sens fort, selon un résultat de Lenstra [48].

Théorème 1. Le problème Π_2 est NP-difficile au sens fort.

Démonstration. D'après la proposition précédente, il suffit de démontrer que résoudre Π'_A se ramène à résoudre Π_2 via une réduction polynomiale.

Soit I une instance de Π'_A.

Les données sont :

- L'ensemble J^0 des tâches,
- l'ensemble M^0 des machines,
- les temps d'exécution p_{ij},
- les priorités ω_i des tâches.

On forme alors l'instance I' de Π_2 :

- L'ensemble des tâches $J = J^0$.
- L'ensemble des machines $M = M^0$.
- L'ensemble des ressources auxiliaires $A = \{A_1, \ldots, A_n\}$.
- Pour toute tâche J_i, $\varphi_i = i$ (chaque tâche a sa propre ressource auxiliaire qu'elle ne partage avec aucune autre tâche d'où la suppression des contraintes concernant les ressources auxiliaires), $\mathcal{M}_i = M$ (plus de contraintes d'éligibilité des machines), $w_i = \omega_i$, $c_i = 0$, et pour toute machine M_j, $\rho_{ij} = p_{ij}$.

– Pour toute ressource A_i, $R_i = 0$, ce qui implique (par le temps unitaire de transfert) que la première tâche sur chaque machine commencera au plus tôt à la date $t = 1$. Ce point précis est la raison pour laquelle on a introduit ce problème.
– Les temps de setup β_{ijk} sont tous nuls.

On vérifie que la réduction est valide, ce qui entraîne le résultat.

En effet, comme Π'_A est équivalent à Π_A, et que ce dernier est NP-difficile au sens fort puisqu'il s'agit d'une généralisation de $P||\sum w_i C_i$, on peut déduire que Π_2 est NP-difficile au sens fort. □

2.3 Maximisation du nombre de plaquettes traitées avant un horizon donné

Tout d'abord, rappelons que le problème Π_B est NP-difficile au sens fort, puisqu'il s'agit d'une généralisation de $P||C_{\max}$ (voir [25]).

Le résultat repose sur la remarque suivante :

Proposition 2. *Soit I une instance de Π_1. On note $C_m(I)$ le makespan minimal pour I. On a la propriété suivante :*

Si $H \geq C_m(I)$, alors la solution optimale est de valeur $\sum_{i=1}^{n} c_i$. Si $H < C_m(I)$, alors elle est de valeur strictement inférieure à $\sum_{i=1}^{n} c_i$.

Démonstration. Si $H \geq C_m(I)$, alors il existe un ordonnancement dont le makespan est inférieur à H et par conséquent dont toutes les tâches se terminent avant H. Ce qui implique que le coût de la solution optimale est le nombre total de plaquettes, à savoir $\sum_{i=1}^{n} c_i$.

Si $H < C_m(I)$, par le caractère optimal de $C_m(I)$, il n'existe pas de solution de makespan inférieur ou égal à H. Pour toute solution, il existe donc au moins une tâche dont l'exécution se termine après H, d'où le résultat. □

On montre dans ce qui suit que si le problème Π_1 peut être résolu en temps polynomial, alors il en est de même du problème dont la fonction objectif est le makespan (à minimiser), problème que l'on notera Π_C, et non le nombre de plaquettes traitées avant H (à maximiser). Cela permettra, avec un raisonnement par l'absurde, d'arriver au résultat qui nous intéresse.

Lemme 1. *S'il existe un algorithme polynomial (resp. pseudo-polynomial) pour* Π_1*, alors il existe un algorithme polynomial (resp. pseudo-polynomial) pour* Π_C*.*

Démonstration. Soit I une instance du problème Π_1 et soient $K_1 = 0$ et K_2 une borne supérieure du makespan minimal pour I. Cette borne supérieure peut se déterminer en construisant une solution avec un algorithme de complexité linéaire, par exemple un algorithme de liste utilisant la règle LPT (Longest Processing Time). On considère un algorithme de résolution polynomial (resp. pseudo-polynomial) pour Π_1. On procède alors comme suit pour déterminer le makespan minimal.

On exécute l'algorithme de résolution polynomiale (resp. pseudo-polynomial) pour Π_1 avec $H = \lfloor \frac{K_1+K_2}{2} \rfloor$. Si la solution optimale est de coût $\sum_{i=1}^{n} c_i$, alors d'après la proposition précédente, H est supérieur ou égal au makespan minimal, ce qui amène à relancer l'opération, avec $K_2 = H$. Sinon, H est strictement inférieur au makespan minimal, donc on relance l'opération avec $K_1 = H$.

On procède ainsi par dichotomie jusqu'à aboutir (lorsque $K_1 = K_2$) à la plus petite valeur de H pour laquelle l'optimum est de coût $\sum_{i=1}^{n} c_i$, c'est-à-dire le makespan.

Le nombre d'itérations étant de l'ordre de $\log K_2$, ce qui est polynomial en fonction de la taille de l'instance (puisque le makespan peut être exprimé comme une fonction exponentielle de la taille de l'instance), l'algorithme utilisé à chaque itération étant de plus supposé polynomial (resp. pseudo-polynomial), on peut ainsi déduire une solution optimale de Π_C en temps polynomial (resp. pseudo-polynomial).

□

Le résultat qui précède permet de terminer un raisonnement par l'absurde, en s'appuyant sur le fait que Π_C est NP-difficile au sens fort, ce qui va être établi par la suite. On utilise notamment le fait que Π_B est NP-difficile au sens fort.

Proposition 3. *Le problème* Π_C *est NP-difficile au sens fort.*

Démonstration. On montre que si l'on peut résoudre toute instance de Π_C en temps polynomial, il en est de même pour Π'_B, ce qui entraîne le résultat puisque Π'_B et Π_B sont équivalents par la proposition 1.

La réduction se fait de façon analogue à celle décrite dans la démonstration du théorème 1.

Il est alors aisé de vérifier que la réduction est valide, d'où le résultat.

□

Théorème 2. Le problème Π_1 est NP-difficile au sens fort.

Démonstration. D'après le lemme 1 et la proposition 1, on déduit immédiatement par l'absurde que le problème Π_1 est NP-difficile au sens fort. □

2.4 Minimiser le nombre de transferts

Dans ce qui suit, on s'intéresse à la minimisation du nombre de déplacements des ressources auxiliaires, c'est-à-dire le nombre de changements de machines pour une ressource auxiliaire. En établissant un lien avec un autre problème d'optimisation combinatoire, le *Set Cover*, on justifie la validité des approches de résolution de ce problème présentées par la suite.

2.4.1 Représentation des solutions

Soit $\Pi_{J,M,A}$ une instance du problème d'ordonnancement avec l'ensemble de tâches J, l'ensemble de machines M et l'ensemble de ressources auxiliaires A.

On note $\chi \subset \{1,\ldots,m\}^n \times \{1,\ldots,n\}^n \times \mathbb{R}_+^n$ l'ensemble des solutions réalisables du problème. On peut représenter une solution du problème de la façon suivante avec le vecteur (x, y, t) dont les composantes se décrivent ainsi :

- Le vecteur $x = (x^1, \ldots, x^\ell)$ où $x^i = (x_1^i, \ldots, x_{n_i}^i)$ et $x_j^i \in \{1,\ldots,m\}$ est l'indice de la machine sur laquelle est exécutée la j-ème tâche (dans l'ordre chronologique d'exécution) nécessitant A_i. Par exemple, supposons que J_3, J_7 et J_4 nécessitent la même ressource auxiliaire A_i et sont exécutées dans l'ordre J_3, J_7, J_4. Elles nécessitent la même ressource auxiliaire et par conséquent ne peuvent pas être exécutées en parallèle. Alors, si J_3 et J_4 sont affectées à M_2, alors que J_7 est affectée à M_1 on a $x_1^i = x_3^i = 2$ et $x_2^i = 1$.
- Le vecteur $y = (y^1, \ldots, y^\ell)$ où $y^i = (y_1^i, \ldots, y_{n_i}^i)$ et $y_j^i \in \{1,\ldots,n\}$ est l'indice de la j-ème tâche (dans l'ordre chronologique d'exécution) nécessitant A_i. En reprenant le même exemple que précédemment, on a $y_1^i = 3$, $y_2^i = 7$ et $y_3^i = 4$.
- Le vecteur $t = (t_1, \ldots, t_n)$ où t_i est la date de début d'exécution de la tâche d'indice i.

Définition 1. Soit $f : \chi \to \mathbb{N}$ qui à toute solution réalisable (x, y, t) associe le nombre de déplacements de ressources auxiliaires induit par cette solution. On a,

pour tout $(x, y, t) \in \chi$:

$$f(x, y, t) = \sum_{i=1}^{\ell} \left| \{ j \in \{0, \ldots, n_i - 1\} | x_j^i \neq x_{j+1}^i \} \right|$$

avec pour tout $i \in \{1, \ldots, \ell\}$, $x_0^i = R_i$.

2.4.2 Simplification du problème

Par définition, on voit que f ne dépend pas du vecteur t. Ainsi, soit $(\hat{x}, \hat{y}, \hat{t})$ une solution réalisable du problème Π. Alors pour tout t tel que $(\hat{x}, \hat{y}, t) \in \chi$, on a $f(\hat{x}, \hat{y}, \hat{t}) = f(\hat{x}, \hat{y}, t)$.

Le résultat qui suit permet de rechercher la solution optimale dans

$$\chi_1 = \{(x, y) \mid \exists t : (x, y, t) \in \chi\} \subset \{1, \ldots, m\}^n \times \{1, \ldots, n\}^n.$$

Proposition 4. *Notons χ_1 la projection de χ sur les composantes de x et de y, c'est-à-dire l'ensemble des vecteurs (x, y) tels qu'il existe t pour lequel $(x, y, t) \in \chi$. Soit $(x, y) \in \chi_1$. On peut alors, à partir de (x, y), déterminer une solution en temps $O(n)$ par l'algorithme* Construire-Ordo(x, y) *où l'on a fixé les valeurs de β_{0ji}, y_0^i, t_0 et ρ_{0j} à 0 :*

```
1  Construire-Ordo(x, y)
2  pour i ← 1 à ℓ faire
3      S[i] ← R_i        ▷ position courante des ressources
4  pour i ← 1 à m faire
5      D[i] ← 0          ▷ dernière tâche ordonnancée sur chaque machine
6  pour i ← 1 à ℓ faire
7      pour j ← 1 à n_i faire
8          τ ← 0
9          si S[i] ≠ x_j^i alors
10             τ ← 1         ▷ temps unitaire de transfert
11         S[i] ← x_j^i      ▷ mise à jour de la position du masque
12         t_{y_j^i} ← max(t_{y_{j-1}^i} + ρ_{y_{j-1}^i x_{j-1}^i} + τ, t_{D[x_j^i]} + ρ_{D[x_j^i]x_j^i} + β_{D[x_j^i]x_j^i y_j^i})
13         D[x_j^i] ← y_j^i  ▷ mise à jour de la dernière tâche exécutée
14 retourner (t_1, ..., t_n)
```

Algorithme 1: Complétion d'une solution

Démonstration. On montre que l'algorithme CONSTRUIRE-ORDO(x, y) détermine une solution réalisable en temps $O(n)$.

- *Complexité :* À chaque itération, des boucles imbriquées s'exécutent en temps constant $O(1)$. Le nombre d'itérations est de $\sum_{i=1}^{\ell} n_i = n$, d'où la complexité de l'algorithme.
- *Validité :* Montrons qu'à la fin de chaque itération des boucles imbriquées, la date de début d'exécution de la tâche y_j^i vérifie :

  1. l'ordre d'exécution des tâches ayant la même ressource auxiliaire requise, induit par la solution (x, y),
  2. et les contraintes du problème (temps de setup, non simultanéité d'exécution de plusieurs tâches sur une même machine, transport unitaire de la ressource associée et non simultanéité d'utilisation d'une ressource).

  Pour démontrer le premier point, on pose D_k l'indice de la dernière tâche ordonnancée sur la machine M_k. Dans l'algorithme, cette valeur est représentée par $D[k]$.
  On vérifie ici que la tâche $J_{y_j^i}$ est toujours exécutée sur la machine $M_{x_j^i}$.
  Par ailleurs, pour tout $i \in \{1, \ldots, \ell\}$ on calcule les dates de début d'exécution des t_j^i dans l'ordre croissant de l'indice j (de 1 à n_i) et la formule

$$t_{y_j^i} = \max(t_{y_{j-1}^i} + \rho_{y_{j-1}^i x_{j-1}^i} + \tau, t_{D_{x_j^i}} + \rho_{D_{x_j^i} x_j^i} + \beta_{D_{x_j^i} x_j^i y_j^i}) \tag{2.1}$$

  (où τ vaut 1 s'il faut un déplacement de la ressource auxiliaire requise, 0 sinon) permet d'écrire l'inégalité :

$$t_{y_j^i} \geq t_{y_{j-1}^i},$$

  ce qui confirme que la solution retournée par l'algorithme vérifie l'ordre d'exécution des tâches nécessitant la même ressource auxiliaire, imposé par (x, y).
  Concernant le second point, les inégalités

$$t_{y_j^i} \geq t_{D_{x_j^i}} + \rho_{D_{x_j^i} x_j^i}$$
$$t_{y_j^i} \geq t_{y_{j-1}^i} + \rho_{y_{j-1}^i x_{j-1}^i} + \tau$$
$$t_{y_j^i} \geq t_{D_{x_j^i}} + \rho_{D_{x_j^i} x_j^i} + \beta_{D_{x_j^i} x_j^i y_j^i}$$

déduites de (2.1), permettent respectivement de valider la satisfaction des contraintes de non-simultanéité de l'exécution de plusieurs tâches sur une même machine, de temps unitaire de transferts de ressources auxiliaires, de non simultanéité de l'utilisation d'une ressource auxiliaire et de temps de setup.

□

La proposition 4 permet de résoudre le problème en restreignant l'ensemble des solutions à χ_1. Ce qui nous permet d'introduire la fonction suivante.

Définition 2. La fonction $\tilde{f} : \chi_1 \to \mathbb{N}$ est définie comme suit : Pour tout $(x, y) \in \chi_1$, $\tilde{f}(x, y) = f(x, y, t)$, où t est tel que $(x, y, t) \in \chi$. $\tilde{f}$ est bien définie car pour tous t et t' tels que (x, y, t) et $(x, y, t') \in \chi$, on a $f(x, y, t) = f(x, y, t')$.

La proposition 4 permet ainsi de considérer le problème :

$$\min_{(x,y)\in\chi_1} \tilde{f}(x, y),$$

qui est équivalent au (i.e. qui a les mêmes solutions optimales que le) problème $\Pi'_{J,M,A}$:

$$\min_{(x,y)\in\chi_1} \tilde{f}(x, y) + \ell,$$

car ℓ est une constante.

Pour tout $i \in \{1, \ldots, \ell\}$, on considère la projection χ_1^i de χ_1 sur les composantes de x^i et y^i ; on introduit alors la fonction $f_i : \chi_1^i \to \mathbb{N}$, définie par

$$f_i(x^i, y^i) = \left|\left\{j \in \{0, \ldots, n_i - 1\} \middle| x_j^i \neq x_{j+1}^i\right\}\right|.$$

Ce qui implique l'égalité

$$\tilde{f}(x, y) + \ell = \sum_{i=1}^{\ell} (f_i(x^i, y^i) + 1).$$

Proposition 5. *L'égalité ci-dessous est vérifiée :*

$$\min_{(x,y)\in\chi_1} \tilde{f}(x, y) + \ell = \sum_{i=1}^{\ell} \min_{(x^i,y^i)\in\chi_1^i} (f_i(x^i, y^i) + 1).$$

et pour toute solution optimale (x^i, y^i), $i \in \{1, \ldots, \ell\}$, de $\Pi'_{E_i,M,\{A_i\}}$, on obtient une solution optimale $(x^1, \ldots, x^\ell, y^1, \ldots, y^\ell)$ de $\Pi'_{J,M,A}$.

Démonstration. On rappelle que pour toute ressource auxiliaire A_i, E_i est l'ensemble des tâches nécessitant A_i. On considère, pour tout $i \in \{1, \ldots, \ell\}$, le problème

$$\Pi'_{E_i,M,\{A_i\}} : \min_{(x^i,y^i)\in\chi_1^i} f_i(x^i, y^i) + 1.$$

Il suffit de montrer que tout ℓ-uplet de solutions $((x^1, y^1), \ldots, (x^\ell, y^\ell)) \in \chi_1^1 \times \ldots \times \chi_1^\ell$ des problèmes $\Pi'_{E_i,M,\{A_i\}}$ fournit une solution $(x^1, \ldots, x^\ell, y^1, \ldots, y^\ell)$ de $\Pi'_{J,M,A}$.

En effet, chacune des tâches de J appartient à un et un seul ensemble parmi les ensembles E_i. Ces derniers forment donc une partition de J, ce qui implique que les contraintes d'exécution unique de chacune des tâches sont vérifiées par toute solution issue de ℓ solutions respectives $((x^1, y^1), \ldots, (x^\ell, y^\ell))$ des problèmes $\Pi'_{E_i,M,\{A_i\}}$. Les contraintes de qualification des machines (puisqu'il s'agit de contraintes de ces problèmes) sont aussi vérifiées.

On peut vérifier que réciproquement, toute solution $(x^1, \ldots, x^\ell, y^1, \ldots, y^\ell)$ fournit ℓ solutions $((x^1, y^1), \ldots, (x^\ell, y^\ell))$ pour les problèmes $\Pi'_{E_i,M,\{A_i\}}$.

Considérons les solutions $((x^{1^*}, y^{1^*}),\ldots,(x^{\ell^*}, y^{\ell^*}))$ optimales pour les problèmes $\Pi'_{E_i,M,\{A_i\}}$. Toute solution optimale

$$(\hat{x}^1, \ldots, \hat{x}^\ell, \hat{y}^1, \ldots, \hat{y}^\ell)$$

de $\Pi'_{J,M,A}$ a pour coût

$$\sum_{i=1}^{\ell}(f_i(\hat{x}^i, \hat{y}^i) + 1).$$

Or, on a :

$$\sum_{i=1}^{\ell}(f_i(x^{i^*}, y^{i^*}) + 1) = \sum_{i=1}^{\ell} \min_{(x^i,y^i)\in\chi_1^i}(f_i(x^i, y^i) + 1) \leq \sum_{i=1}^{\ell}(f_i(\hat{x}^i, \hat{y}^i) + 1).$$

$(x^{1^*}, \ldots, x^{\ell^*}, y^{1^*}, \ldots, y^{\ell^*})$ étant une solution réalisable de $\Pi'_{J,M,A}$, elle est optimale.

Soit $(x^{1^*}, \ldots, x^{\ell^*}, y^{1^*}, \ldots, y^{\ell^*})$ une solution optimale de $\Pi'_{J,M,A}$. S'il existe un couple (x^{k^*}, y^{k^*}) tel que $f_k(x^{k^*}, y^{k^*}) + 1 \neq \min_{(x^k,y^k)\in\chi_1^k} f_k(x^k, y^k) + 1$, alors

$$\tilde{f}(x^*, y^*) + \ell = \sum_{i=1}^{\ell}(f_i(x^{i^*}, y^{i^*}) + 1) < \sum_{i=1}^{\ell} \min_{(x^i,y^i)\in\chi_1^i}(f_i(x^i, y^i) + 1)$$

d'où la contradiction et le résultat. □

Corollaire 1. *Résoudre le problème* $\Pi'_{J,M,A}$ *revient à résoudre les* ℓ *problèmes*

$$\Pi'_{E_1,M,\{A_1\}}, \ldots, \Pi'_{E_\ell,M,\{A_\ell\}}$$

et déterminer une solution en $O(\ell)$.

Démonstration. Il suffit de résoudre les problèmes $\Pi'_{E_i,M,\{A_i\}}$ et déterminer le vecteur solution à partir des solutions optimales des sous-problèmes en $O(\ell)$.

□

Le schéma général de la résolution du problème est donné par l'algorithme 2, MIN-DÉPLACEMENTS-MASQUES($\Pi'_{J,M,A}$).

```
1 MIN-DÉPLACEMENTS-MASQUES(Π'_{J,M,A})
2 pour i ← 1 à ℓ faire
3     (x^i, y^i) ← MIN-MASQUES(Π'_{E_i,M,{A_i}})
4 retourner (x^1, ..., x^ℓ, y^1, ..., y^ℓ)
```

Algorithme 2: Détermination d'une solution de $\Pi'_{J,M,A}$

2.4.3 Réduction polynomiale

Il reste à concevoir un algorithme MIN-MASQUES pour résoudre le problème $\Pi'_{E_i,M,\{A_i\}}$.

On définit la notion de *bloc* :

Définition 3. Soit $x = (x_1, \ldots, x_n)$ un vecteur de n composantes. On appelle *bloc* de x un vecteur $(x_i, x_{i+1}, \ldots, x_{i+k})$ de composantes de x telles que :

- $x_i = x_{i+1} = \ldots = x_{i+k}$
- si $i > 1$, $x_i \neq x_{i-1}$
- si $i + k < n$, $x_{i+k} \neq x_{i+k+1}$.

Ici, un bloc va correspondre à une suite de tâches nécessitant le même masque et exécutées sur la même machine. La proposition suivante permet de réduire l'ensemble de recherche.

Proposition 6 (Propriété de dominance)**.** *On rappelle qu'on considère pour toute solution* (x^i, y^i)*, les valeurs* $x^i_0 = R_i$*. Toute solution* $(x^i_1, \ldots, x^i_{n_i}, y^i_1, \ldots, y^i_{n_i})$ *de* $\Pi'_{E_i,M,\{A_i\}}$ *vérifiant la propriété suivante est dominante :*

Pour tout $j \in \{0, \ldots, n_i - 1\}$ *tel que* $x^i_j \neq x^i_{j+1}$, *on a* $x^i_j \neq x^i_k$ *pour tout* $k \in \{j+2, \ldots, n_i\}$.

Démonstration. Soit $(x^i, y^i) = (x^i_1, \ldots, x^i_{n_i}, y^i_1, \ldots, y^i_{n_i})$ une solution de $\Pi'_{E_i, M, \{A_i\}}$ ne vérifiant pas cette propriété. Soit alors $j \in \{0, \ldots, n_1 - 1\}$ l'indice tel que $x^i_j \neq x^i_{j+1}$ et qu'il existe $k \in \{j+2, \ldots, n_i\}$ tel que $x^i_j = x^i_k$.

Soit $k' = \min\{k \in \{j+2, \ldots, n_i\} \mid x^i_j = x^i_k\}$. k' existe par hypothèse.

Considérons alors la solution

$$(x'^i, y'^i) = (x^i_1, \ldots, x^i_j, x^i_{k'}, x^i_{j+1}, \ldots, x^i_{k'-1}, x^i_{k'+1}, \ldots, x^i_{n_i}, y^i_1, \ldots, y^i_j, y^i_{k'}, y^i_{j+1}, \ldots, y^i_{k'-1}, y^i_{k'+1}, \ldots, y^i_{n_i}).$$

On a donc

$$f_i(x'^i, y'^i) = \begin{cases} f_i(x^i, y^i) - 1 & \text{si } k' = n_i \\ f_i(x^i, y^i) & \text{si } k' < n_i \text{ et } x^i_{k'} = x^i_{k'+1} \\ f_i(x^i, y^i) - 2 & \text{si } k' < n_i,\ x^i_{k'} \neq x^i_{k'+1} \text{ et } x^i_{k'-1} = x^i_{k'+1} \\ f_i(x^i, y^i) - 1 & \text{sinon.} \end{cases}$$

d'où $f_i(x'^i, y'^i) \leq f_i(x^i, y^i)$ et le résultat.

On peut par ailleurs démontrer qu'il existe une solution dominante dont le coût est strictement inférieur à celui de (x^i, y^i). En effet, supposons que l'on soit dans le cas où $k' < n_i$ et $x^i_{k'} = x^i_{k'+1}$; le coût est donc le même. On peut alors appliquer le même procédé à la solution (x'^i, y'^i) jusqu'à aboutir à l'un des autres cas. En effet, on peut répéter ce procédé puisque l'indice $k' + 1$ est l'indice minimal de $\{j+2, \ldots, n_i\}$ tel que $x'^i_j = x^i_{k'+1}$.

□

Corollaire 2. *Déterminer une solution optimale parmi les solutions dominantes décrites précédemment revient à déterminer le nombre minimal de machines par lesquelles passe la ressource auxiliaire* A_i.

Cela entraîne immédiatement qu'un majorant du nombre minimal de transferts est $m\ell$, *où* m *est le nombre de machines et* ℓ *le nombre de ressources auxiliaires.*

Démonstration. Dans les solutions dominantes (x^i, y^i) décrites dans la proposition 6, le nombre de blocs de $(x^i_0, x^i_1, \ldots, x^i_{n_i})$ est égal au nombre de machines

par lesquelles passe la ressource A_i. Or, le nombre de blocs est égal au nombre de déplacements de A_i augmenté d'une unité : $f_i(x^i, y^i) + 1$, fonction objectif de $\Pi'_{E_i,M,\{A_i\}}$. Ainsi, parmi ces solutions dominantes, minimiser le nombre de blocs revient à minimiser le nombre de machines par lesquelles passe A_i. □

Le corollaire 2 montre que résoudre le problème $\Pi'_{E_i,M,\{A_i\}}$ revient à déterminer le nombre minimal de machines par lesquelles passe A_i.

Le théorème suivant permet de se ramener à la résolution d'un autre problème d'optimisation combinatoire très étudié dans la littérature.

Théorème 3. Le problème de couverture minimale par des ensembles (*Set Covering*) se réduit à un problème $\Pi'_{E_i,M,\{A_i\}}$, $i \in \{1, \ldots, \ell\}$.

Démonstration. Le problème de couverture minimale par des ensembles a les données suivantes :

- Un ensemble T (l'ensemble à couvrir).
- s ensembles $T_1, \ldots, T_s$ (ensembles couvrants) inclus dans T tels que

$$T_1 \cup \ldots \cup T_s = T.$$

L'objectif est de déterminer un sous-ensemble $\{V_1, \ldots, V_k\}$ de $\{T_1, \ldots, T_s\}$ de cardinalité minimale tel que $\cup_{i=1}^{k} V_i = T$.

Soit une instance du problème de couverture minimale par des ensembles. Montrons comment construire une instance du problème $\Pi'_{E_i,M,\{A_i\}}$.

- $E_i = T$.
- $M_j(i) = T_j$, pour tout $1 \leq j \leq m$, et $R_i = 0$.

La solution optimale de cette instance du problème fournit le nombre minimal de machines pouvant exécuter les tâches de E_i, donc le nombre minimal de machines sur lesquelles passe la ressource auxiliaire A_i. On obtient de cette façon la couverture minimale de l'ensemble T par les sous-ensembles $T_1, \ldots, T_s$.

Soit $\{M_{j_1}(i), \ldots, M_{j_d}(i)\}$ une solution optimale du problème. La propriété de dominance impose l'exécution de toutes les tâches de $M_{j_1}(i)$ sur M_{j_1}, puis de toutes les tâches de $M_{j_2}(i) \setminus M_{j_1}(i)$ sur M_{j_2}, et ainsi de suite jusqu'à toutes les tâches de $M_{j_d}(i) \setminus \left(\cup_{r=1}^{d-1} M_{j_r}(i)\right)$ sur $M_{j_d}(i)$, dans cet ordre. Ceci donne le nombre minimal de sous-ensembles couvrant $T = E_i$, sinon on pourrait proposer un ensemble de cardinalité inférieure comme solution du problème, d'où un nombre inférieur de machines par lesquelles passe A_i, ce qui est absurde.

Réciproquement, toute solution optimale du problème de couverture minimale par des ensembles correspond à une solution optimale du problème $\Pi'_{E_i,M,\{A_i\}}$. Le même raisonnement permet de conclure.

□

Proposition 7. *Il existe une réduction polynomiale de $\Pi'_{E_i,M,\{A_i\}}$ au problème de couverture minimale par des ensembles.*

Démonstration. Soit I une instance de $\Pi'_{E_i,M,\{A_i\}}$; on construit une instance I' du problème de couverture minimale par des ensembles comme suit :

L'ensemble à couvrir T est égal à $E_i \cup \{J_0\}$ où J_0 est une tâche fictive, et les ensembles couvrants sont les ensembles $M_j(i)$ $(0 \leq j \leq m)$ où $M_{R_i}(i)$ contient en plus la tâche J_0. Ceci garantit que la solution inclura l'emplacement initial de A_i.

Toute solution optimale de I' correspond à une solution optimale de $\Pi'_{E_i,M,\{A_i\}}$ et réciproquement. □

2.4.4 Complexité du problème

Théorème 4 (Complexité du problème $\Pi'_{J,M,A}$)**.** Le problème $\Pi'_{J,M,A}$ est NP-difficile au sens fort.

Démonstration. Considérons le cas particulier du problème où $\ell = 1$ et $R_1 = M_0$. Montrer que ce problème est NP-difficile au sens fort entraîne le résultat.

Pour ce faire, on peut réduire le problème de couverture minimale au problème $\Pi'_{J,M,A}$. Soit $I = (E, \{M_1, \ldots, M_m\})$ une instance du problème de couverture minimale par des ensembles.

On obtient une instance I' de $\Pi'_{J,M,A}$ en posant $J = E$ et pour tout $j \in \{1, \ldots, m\}$, $Q_j = M_j$. D'après le corollaire 2, toute solution optimale de I fournit le nombre minimal de machines par lesquelles passe la ressource auxiliaire A_1. Ainsi, la solution optimale (x^*, y^*) de I' se déduit de la solution optimale de I, de coût $\tilde{f}(x^*, y^*) = (\tilde{f}(x^*, y^*) + 1) - 1$.

En effet, $\tilde{f}(x^*, y^*)$ est le nombre de machines, hormis M_0 par lesquelles passe A_1. C'est donc le nombre total de machines par lesquelles passe A_1, diminué d'une unité.

On a ainsi effectué une réduction polynomiale d'un problème NP-difficile au sens fort à un problème équivalent (celui où la fonction objectif est $\tilde{f} - 1$) à un cas particulier de $\Pi'_{J,M,A}$. D'où le résultat.

□

Remarque 1. Le résultat précédent est vrai lorsque le nombre m de machines n'est pas fixé, c'est-à-dire que c'est un paramètre du problème. Le résultat qui suit analyse la complexité du cas où m est fixé.

Théorème 5. Lorsque m est une valeur constante, le problème $\Pi'_{J,M,A}$ est dans P.

Démonstration. Démontrer ce théorème revient, d'après le corollaire 1 et la proposition 7, à démontrer que le problème de couverture minimale par des ensembles est dans P lorsque le nombre d'ensembles couvrants est une constante. Or, le nombre de solutions du problème est le nombre de parties de l'ensemble des m sous-ensembles couvrants, c'est-à-dire 2^m. On déduit de la proposition 7 qu'il y a $\ell \times 2^m$ solutions à une instance du problème $\Pi'_{J,M,A}$ parmi lesquelles on peut rechercher la solution optimale. La complexité est de $O(\ell)$, puisque $C = 2^m$ est supposé constant. D'où le résultat.

□

2.5 Conclusion

Ce chapitre a présenté trois résultats de complexité :

1. La minimisation du critère $\sum_j w_j C_j$ est NP-difficile au sens fort.
2. La maximisation du nombre de plaquettes traitées dans un horizon H de temps fixé est NP-difficile au sens fort ; en effet on montre que ce problème est au moins aussi difficile que la minimisation du makespan.
3. La minimisation du nombre de transferts de ressources auxiliaires est
   – NP-difficile au sens fort si le nombre de machines est une donnée du problème, puisqu'il est au moins aussi difficile que le problème de *Set Covering* (couverture par des ensembles),
   – dans P si le nombre de machines est constant. Dans ce cas, une simple méthode d'énumération fournit la solution optimale.

Ces théorèmes amènent naturellement à aborder des problèmes de taille industrielle avec des méthodes non exactes (métaheuristique, heuristique gloutonne, algorithmes d'approximation, etc.). Les méthodes de résolutions étudiées pour ces trois problèmes sont présentées dans les deux chapitres suivants et sont motivées par les propriétés de ces problèmes et l'étude de complexité.

Chapitre 3

Résolution exacte des problèmes d'ordonnancement monocritère

Dans ce chapitre, la modélisation des trois problèmes d'ordonnancement étudiés au chapitre 2 au moyen de la programmation linéaire en nombres entiers est exposée. Il s'agit, pour deux d'entre eux, de modèles à variables indexées par le temps, qui nécessitent un grand nombre de variables. Des inégalités valides ajoutées pour renforcer les formulations sont proposées. Une heuristique primale permet dans certains cas d'obtenir des solutions réalisables au cours de l'exploration arborescente de l'espace des solutions. La dernière partie du chapitre présente les performances des formulations et l'impact des inégalités valides sur la borne obtenue par relaxation linéaire et sur la vitesse de résolution sur des instances générées aléatoirement.

3.1 Rappel de la problématique

Le problème étudié est un problème d'ordonnancement sur machines parallèles différentes avec des contraintes motivées par l'atelier de photolithographie en fabrication de semi-conducteurs. C'est un problème avec ensembles de machines qualifiées (chaque tâche ne peut être exécutée que par un sous-ensemble donné de machines) et temps de setup dépendant de la séquence et de la machine. Par ailleurs, à chaque tâche est associée une ressource auxiliaire. Ainsi, deux tâches associées à la même ressource ne peuvent s'exécuter en parallèle. Enfin, le déplacement d'une ressource entre deux machines différentes requiert une durée unitaire de transfert.

Le nombre de plaquettes produites dans un horizon fixé H est à maximiser et la somme pondérée des dates de fin ($\sum_j w_j C_j$) est à minimiser. Le nombre de transferts de ressources auxiliaires entre les machines, à minimiser, est un troisième critère, non étudié dans la littérature.

On rappelle les données du problème :

- Un ensemble de n tâches, ou *jobs* $J = \{J_1, \ldots, J_n\}$.
- Un ensemble de m machines $M = \{M_1, \ldots, M_m\}$.

- Un ensemble de ℓ ressources auxiliaires $A = \{A_1, \ldots, A_\ell\}$, $\ell \leq n$.
- Pour toute tâche J_i, son nombre de *plaquettes* $c_i \in \mathbb{N}$, sa priorité w_i, son indice de ressource requise $\varphi_i \in \{1, \ldots, \ell\}$ et son ensemble d'indices de machines qualifiées $\mathcal{M}_i \subset \{1, \ldots, m\}$.
- Pour toute tâche J_i et toute machine M_j telle que $j \in \mathcal{M}_i$, le temps d'exécution $\rho_{ij} \in \mathbb{N}$ de J_i par M_j.
- Pour tout couple de tâches (J_i, J_k) et toute machine M_j, $j \in \mathcal{M}_i \cap \mathcal{M}_k$, le temps de setup $\beta_{ijk} \in \mathbb{N}$ mis par la machine M_j pour exécuter J_k juste après J_i.
- L'emplacement initial $R_i \in \{0, \ldots, m\}$ de chaque ressource auxiliaire A_i, indice de la machine sur laquelle se trouve initialement la ressource A_i, 0 représentant le dépôt (ou stock) des ressources auxiliaires.

On ajoute les notations suivantes :

- Pour toute ressource auxiliaire $A_i \in A$, on note n_i le nombre de tâches de J la nécessitant et $E_i = \{J_{i_1}, \ldots, J_{i_{n_i}}\}$ l'ensemble de ces tâches :

$$E_i = \{J_k \in J \mid \varphi_k = i\}.$$

- Pour toute machine $M_j \in M$, l'ensemble Q_j des tâches pour lesquels M_j est qualifiée.
- Pour tout machine $M_j \in M$, pour toute ressource auxiliaire $A_i \in A$, l'ensemble

$$M_j(i) = \{k \mid J_k \in Q_j \cap E_i\}$$

des indices des tâches de E_i pour lesquelles M_j est qualifiée.
On pose de plus $M_0(i) = \emptyset$ pour tout i, $1 \leq i \leq \ell$.

3.2 Maximiser le nombre de plaquettes avant H et minimiser $\sum_j w_j C_j$

Soit Π_1 (resp. Π_2) le problème consistant à maximiser le nombre de plaquettes produites avant H (resp. minimiser la somme pondérée des dates de fin d'exécution).

Soit T un entier suffisamment grand, c'est-à-dire supérieur ou égal au *makespan* d'une solution optimale du problème Π_2 et au *makespan* d'une solution optimale du problème Π_1.

On peut modéliser le problème en discrétisant le temps et en associant une variable binaire u_{ijt} à tout triplet (i, j, t) tel que $i \in \{1, \ldots, n\}, j \in \mathcal{M}_i$ et $t \in \{0, \ldots, T - \rho_{ij}\}$. La variable u_{ijt} est égale à 1 si la tâche i est affectée à la machine j et commence son exécution au temps t et est égale à 0 sinon.

Soit $\mathcal{U}$ l'ensemble des vecteurs $u_{ijt} : \mathcal{U} = \{0,1\}^K$, où $K = \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} (T - \rho_{ij} + 1)$.

Proposition 8. *Il existe une solution optimale de* Π_1 *(resp. de* Π_2*) pouvant être représentée par un élément de* $\mathcal{U}$*.*

Démonstration. Toute solution réalisable de Π_1 (donc de Π_2) implique l'exécution de toutes les tâches, chacune sur l'une de ses machines qualifiées. De plus, T peut être choisi de sorte qu'il existe une solution optimale de Π_1 (et de Π_2) dont le makespan est inférieur ou égal à T. Par exemple, $T = n \times \max_{(i,j) \in \{1,\ldots,n\} \times \mathcal{M}_i} \rho_{ij} + (n-1) \times \max_{(i,k,j) \in \{1,\ldots,n\}^2 \times (\mathcal{M}_i \cup \mathcal{M}_k)} \beta_{ijk}$.

Ainsi, il existe une solution optimale de Π_1 (resp. de Π_2) dans laquelle toute tâche J_i est exécutée sur une machine M_j, $j \in \mathcal{M}_i$, à partir d'une date t telle que $t + \rho_{ij} \leq T$, donc $t \leq T - \rho_{ij}$.

Ce qui implique qu'il existe un élément $(u_{ijt})_{i \in \{1,\ldots,n\}, j \in \mathcal{M}_i, t \in \{0,\ldots,T-\rho_{ij}\}}$ de $\mathcal{U}$ représentant une solution optimale de Π_1 (resp. de Π_2).

□

3.2.1 Programme Linéaire en Nombres Entiers pour maximiser le nombre de plaquettes

Le programme linéaire en nombres entiers suivant (FIT-1) est valide pour le problème Π_1.

$$\text{Max} \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right) \tag{3.1}$$

s. c.

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1 \qquad \forall i = 1, \ldots, n \tag{3.2}$$

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i \tag{3.3}$$

$$u_{ijt} + \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{jii_0})} u_{i_0 j t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\} \tag{3.4}$$

$$\sum_{i=1}^{n} \sum_{\substack{j \in \mathcal{M}_i \\ j \neq R_{\varphi_i}}} u_{ij0} = 0 \tag{3.5}$$

$$u_{ijt} \in \{0,1\} \qquad \forall i = 1, \ldots, n, j \in \mathcal{M}_i, t = 0, \ldots, T-\rho_{ij}. \tag{3.6}$$

La fonction objectif proposée (3.1) est bien l'expression du nombre de plaquettes traitées avant l'horizon H. Les contraintes (3.2) expriment le fait qu'une tâche s'exécute une et une seule fois sur une et une seule de ses machines qualifiées. Les contraintes (3.3) imposent que deux tâches nécessitant la même ressource auxiliaire ne peuvent pas s'exécuter simultanément, tandis que les contraintes (3.4) interdisent que deux tâches ne s'exécutent simultanément sur une même machine, en tenant compte des temps de setup. La contrainte (3.5) exprime le fait qu'une tâche ne peut pas commencer à la date d'origine $t = 0$ sur une machine si le masque requis ne s'y trouve pas initialement.

Nous explicitons par la suite en détail comment a été construit le modèle (FIT-1), et en particulier les contraintes (3.3) et (3.4).

Fonction objectif (3.1)

Proposition 9. *La fonction objectif du problème* Π_1*, à savoir le nombre de pla-*

quettes produites avant H, s'exprime de la façon suivante :

$$\sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right)$$

Démonstration. Selon les notations du chapitre 1, la fonction objectif s'écrit : $\sum_{i=1}^{n} c_i \theta_i$, où

$$\theta_i = \begin{cases} \frac{\min(t_i+\rho_{im_i},H)-t_i}{\rho_{im_i}} & \text{si } t_i \leq H \\ 0 & \text{sinon.} \end{cases}$$

Soit $J_i \in J$ une tâche. Montrer la proposition revient à prouver l'égalité

$$\theta_i = \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} \frac{H-t}{\rho_{ij}} u_{ijt} \right).$$

Si $t_i > H$, pour tout $(j,t) \in \mathcal{M}_i \times \{0,\ldots,H\}$, $u_{ijt} = 0$ donc

$$\theta_i = \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} \frac{H-t}{\rho_{ij}} u_{ijt} \right) = 0.$$

Si $t_i \leq H$, alors si $t_i \leq H - \rho_{ij}$, on a $\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{H-\rho_{ij}} u_{ijt} = 1$ et $u_{ijt} = 0$ pour tous $j \in \mathcal{M}_i$, $t \in \{H-\rho_{ij}+1,\ldots,H\}$ d'où l'égalité

$$\theta_i = \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} \frac{H-t}{\rho_{ij}} u_{ijt} \right) = 1.$$

Si $t_i > H - \rho_{ij}$, alors $\theta_i = \frac{H-t_i}{\rho_{ij}}$ et :
$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{H-\rho_{ij}} u_{ijt} = 0$, $\sum_{j \in \mathcal{M}_i} \sum_{t=H-\rho_{ij}+1}^{H} \frac{H-t}{\rho_{ij}} u_{ijt} = \frac{H-t_i}{\rho_{ij}}$, d'où le résultat. □

Contraintes (3.3) et (3.4) Notons que pour représenter une solution réalisable de Π_1 et Π_2, les éléments $(u_{ijt}) \in \mathcal{U}$ doivent vérifier

$$M'(1-u_{ijt}) \geq \sum_{\substack{i_0 | j \in \mathcal{M}_{i_0} \\ i_0 \neq i}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{jii_0})} u_{i_0 j t_0} + \sum_{\substack{i_0 | \varphi_{i_0}=\varphi_i \\ i_0 \neq i}} \sum_{\substack{j_0 \in \mathcal{M}_{i_0} \\ j_0 \neq j}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} \tag{3.7}$$

pour tout $i \in \{1, \ldots, n\}$, pour tout $j \in \mathcal{M}_i$, pour tout $t \in \{0, \ldots, T - \rho_{ij}\}$, avec M' un majorant du nombre de termes dans la somme du membre de droite.

En effet, soient une tâche J_i, une machine M_j, $j \in \mathcal{M}_i$ et $t \in \{0, \ldots, T - \rho_{ij}\}$. On considère une solution réalisable (u_{ijt}) de Π_1. La contrainte imposant le fait qu'une machine ne peut exécuter qu'une tâche à la fois implique :

$$M'_1(1 - u_{ijt}) \geq \sum_{\substack{i_0 | j \in \mathcal{M}_{i_0} \\ i_0 \neq i}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1)} u_{i_0 j t_0} \tag{3.8}$$

avec M'_1 un majorant du nombre de termes dans la somme du membre de droite.

En effet, on voit que si $u_{ijt} = 1$, la contrainte (3.8) impose qu'aucune autre tâche J_{i_0} ($i_0 \neq i$) ne peut être exécutée sur M_j à partir d'une date située dans l'ensemble $\{t, \ldots, t + \rho_{ij} - 1\}$. Si une somme de variables binaires vaut 0, alors tous les termes de cette somme sont nuls.

Si $u_{ijt} = 0$, la valeur de M'_1 est choisie de sorte que cette inégalité n'est pas contraignante (elle est redondante avec le fait que toutes les valeurs des $u_{i_0 j t_0}$ sont au plus égales à 1).

L'expression $\min(T - \rho_{i_0 j}, t + \rho_{ij} - 1)$ sert simplement à éviter d'inclure dans la somme du membre de droite une variable $u_{i_0 j t_0}$ telle que $t_0 > T - \rho_{i_0 j}$.

De plus, par

$$M'_2(1 - u_{ijt}) \geq \sum_{\substack{i_0 | j \in \mathcal{M}_{i_0} \\ i_0 \neq i}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{iji_0})} u_{i_0 j t_0}$$

(où M'_2 est un majorant du nombre de termes dans la somme du membre de droite) on inclut la contrainte liée aux temps de setup entre l'exécution de deux tâches consécutives sur la même machine.

Par ailleurs, on peut écrire

$$M'_3(1 - u_{ijt}) \geq \sum_{\substack{i_0 | \varphi_{i_0} = \varphi_i \\ i_0 \neq i}} \sum_{\substack{j_0 \in \mathcal{M}_{i_0} \\ j_0 \neq j}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0}$$

(où M'_3 est un majorant du nombre de termes dans la somme du membre de droite), pour vérifier que, si $u_{ijt} = 1$ (i.e. si J_i est exécutée sur la machine M_j à la date t), aucune autre tâche J_{i_0} ($i_0 \neq i$) nécessitant la même ressource ($\varphi_{i_0} = \varphi_i$) ne peut être exécutée sur une autre machine M_{j_0} qualifiée ($j_0 \in \mathcal{M}_{i_0}$) pendant l'exécution de J_i sur M_j, c'est-à-dire dans $\{t, \ldots, t + \rho_{ij} - 1 + 1\}$ (on ajoute 1 à $t + \rho_{ij} - 1$ selon

la contrainte qui impose un temps de setup unitaire pour le déplacement d'une ressource d'une machine à une autre).

La somme des deux inégalités précédentes permet de retrouver l'inégalité (3.7).

Remarque 2. Les égalités (3.2) impliquent que l'on peut poser

$$M' = |Q_j \cup E_{\varphi_i}| - 1.$$

Cela correspond à

$$\begin{aligned}
&|\{i_0 \in \{1,\dots,n\},\, i_0 \neq i,\, j \in \mathcal{M}_{i_0}\} \cup \{i_0 \in \{1,\dots,n\},\, i_0 \neq i,\, \varphi_{i_0} = \varphi_i\}| \\
=&(|Q_j| - 1) + (|E_{\varphi_i}| - 1) - |\{i_0 \in \{1,\dots,n\},\, i_0 \neq i,\, j \in \mathcal{M}_{i_0},\, \varphi_{i_0} = \varphi_i\}| \\
=&|Q_j| + |E_{\varphi_i}| - 2 - (|M_j(\varphi_i)| - 1) \\
=&|Q_j| + |E_{\varphi_i}| - |M_j(\varphi_i)| - 1.
\end{aligned}$$

Remarque 3. On peut réécrire l'inégalité (3.7) sans utiliser le coefficient M' (*big-M*) :

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1 \quad (3.9)$$

$$\forall i = 1,\dots,n, \forall j \in \mathcal{M}_i, \forall t = 0,\dots,T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i$$

et

$$u_{ijt} + \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{ji i_0})} u_{i_0 j t_0} \leq 1 \quad (3.10)$$

$$\forall i = 1,\dots,n, \forall j \in \mathcal{M}_i, \forall t = 0,\dots,T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\}.$$

On retrouve alors les contraintes (3.3) et (3.4). L'intérêt ici est de s'affranchir d'une difficulté de taille : la détermination d'une valeur pertinente pour le coefficient M' pour la relaxation linéaire. Le modèle mathématique comprend néanmoins plus de contraintes.

La contrainte (3.5) impose un temps *unitaire* de déplacement pour les ressources auxiliaires. Elle implique qu'on ne peut pas utiliser une ressource auxiliaire sur une machine à $t = 0$, à moins que la ressource auxiliaire soit déjà sur la machine. Ainsi, une tâche J_i commence son exécution sur une machine qualifiée M_j ($j \in \mathcal{M}_i$) à la date $t = 0$ seulement si $R_{\varphi_i} = j$. Une somme d'entiers dans {0,1} est nulle si et seulement si tous les entiers de la somme sont nuls.

Avec les inégalités précédemment décrites, on satisfait toutes les contraintes des problèmes Π_1 et Π_2. Si l'on peut exprimer la fonction objectif de ces problèmes comme une expression linéaire en fonction des variables u_{ijt} ($i \in \{1, \ldots, n\}, j \in \mathcal{M}_i, t \in \{0, \ldots, T - \rho_{ij}\}$), alors nous avons un programme linéaire en nombres entiers (PLNE).

Remarque 4. Pour le problème Π_1, comme seules les tâches dont l'exécution commence avant H influent sur le coût de la solution, on peut considérer une version du problème dans laquelle la contrainte d'exécution de toutes les tâches est relâchée (les contraintes (3.2) deviennent $\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} \leq 1$). Il convient d'écrire la contrainte comme suit :

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{H} u_{ijt} \leq 1. \tag{3.11}$$

En effet, toute solution optimale de Π_1 est optimale pour la version relâchée évoquée. De plus, toute solution optimale de cette version relâchée, pour laquelle on ordonnance les tâches manquantes aléatoirement sans modifier l'ordonnancement des autres tâches, est optimale pour Π_1 (puisque ces tâches ne changent pas le coût de la solution). On vérifie ce fait par l'absurde.

Cela se ramène donc à résoudre Π_1 par le PLNE (FIT-1) dans lequel les contraintes garantissant l'exécution de toutes les tâches sont remplacées comme décrit ci-dessus. La solution optimale obtenue peut donc être complétée par des choix qui seront de toute façon insignifiants au sens de la fonction objectif de Π_1.

3.2.2 Programme Linéaire en Nombres Entiers pour minimiser la somme des dates de fin

Le programme linéaire en nombres entiers suivant (FIT-2) est valide pour le problème Π_2.

$$\text{Min} \sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right) \tag{3.12}$$

s. c.

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1 \qquad \forall i = 1, \ldots, n \tag{3.13}$$

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i \tag{3.14}$$

$$u_{ijt} + \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{iji_0})} u_{i_0 j t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\} \tag{3.15}$$

$$\sum_{i=1}^{n} \sum_{\substack{j \in \mathcal{M}_i \\ j \neq R_{\varphi_i}}} u_{ij0} = 0 \tag{3.16}$$

$$u_{ijt} \in \{0,1\} \qquad \forall i = 1, \ldots, n, j \in \mathcal{M}_i, t = 0, \ldots, T-\rho_{ij}. \tag{3.17}$$

Proposition 10. *La fonction objectif du problème* Π_2*, s'exprime de la façon suivante :*

$$\sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right).$$

Démonstration. En notant C_i la date à laquelle J_i termine son exécution, la fonction objectif s'écrit :

$$\sum_{i=1}^{n} w_i C_i.$$

Soit $J_i \in J$ une tâche. Il s'agit de prouver que

$$C_i = \sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}).$$

D'après les contraintes (3.2), il existe un unique couple $(j,t) \in \mathcal{M}_i \times \{0, \ldots, T-\rho_{ij}\}$ tel que $u_{ijt} = 1$. Soit (j_0, t_0) ce couple. On a alors : $C_i = t_0 + \rho_{ij_0}$, d'où le résultat. □

Les contraintes observées dans le modèle sont les mêmes que dans la formulation (FIT-1).

3.2.3 Généralisation des Programmes Linéaires en Nombres Entiers

Ces formulations présentent l'avantage de s'adapter facilement à différentes généralisations du problème :

1. Si on considère que les temps de transport des ressources auxiliaires dépendent de l'origine et de la destination (qui, on le rappelle, peuvent être des machines ou la zone de stockage) au lieu d'être constants, on a alors une matrice $(T_{ij})_{i,j\in\{0,\dots,m\}}$ des temps de transport entre les machines, l'indice 0 désignant la zone de stockage. Dans ce cas, la contrainte (3.14) du PLNE (FIT-1) se réécrit :

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0}\setminus\{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0j_0},\, t+\rho_{ij}-1+T_{jj_0})} u_{i_0j_0t_0} + \sum_{j_0 \in \mathcal{M}_{i_0}\cap\{j\}} \sum_{t_0=t}^{\min(T-\rho_{i_0j_0},\, t+\rho_{ij}-1)} u_{i_0j_0t_0} \leq 1$$

   pour tous $i \in \{1,\dots,n\}$, $j \in \mathcal{M}_i$, $t \in \{0,\dots,T-\rho_{ij}\}$ et $i_0 \neq i$ tels que $\varphi_{i_0} = \varphi_i$. En effet, l'ajout de T_{jj_0} à la place de 1 permet de garantir que dans toute solution réalisable, la propriété suivante est vérifiée : si une ressource est utilisée par une tâche J_i sur une machine M_j à partir de la date t, alors elle ne sera pas présente sur une autre machine M_{j_0} dans les $\rho_{ij} - 1 + T_{jj_0}$ unités de temps suivantes ; en effet, cela correspond à la durée d'exécution de J_i ainsi qu'au temps de transport de la ressource auxiliaire en question de la machine M_j à la machine M_{j_0}.
   De plus, la contrainte(3.5) peut être réécrite :

$$\sum_{i=1}^{n} \sum_{\substack{j\in\mathcal{M}_i \\ j\neq R_{\varphi_i}}} \sum_{t=0}^{\min(T_{R_{\varphi_i}j}-1,\, T-\rho_{ij})} u_{ijt} = 0.$$

2. Au lieu d'avoir une quantité constante de ressources auxiliaires d'un même type, on peut considérer que, pour tout type, il existe une quantité initiale donnée, qui peut différer de 1. On note alors Γ_i $(1 \leq i \leq \ell)$ la quantité de ressources auxiliaires du type A_i. Par exemple, si l'on pose $\Gamma_5 = 3$, cela signifie qu'il existe 3 ressources du type de A_5. Ceci implique qu'on peut exécuter en parallèle au plus 3 tâches nécessitant la ressource A_5. Cette

nouvelle donnée s'intègre aussi aisément dans la formulation proposée. La contrainte (3.7) se scinde en deux contraintes distinctes :

$$M'(1-u_{ijt}) \geq \sum_{\substack{i_0 | j \in \mathcal{M}_{i_0} \\ i_0 \neq i}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{iji_0})} u_{i_0 j t_0} \tag{3.18}$$

et

$$M'(1-u_{ijt})+(\mathbf{\Gamma}_{\varphi_\mathbf{i}}-\mathbf{1}) \geq \sum_{\substack{i_0 | \varphi_{i_0}=\varphi_i \\ i_0 \neq i}} \sum_{\substack{j_0 \in \mathcal{M}_{i_0} \\ j_0 \neq j}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} \tag{3.19}$$

pour tous $i \in \{1,\ldots,n\}$, $j \in \mathcal{M}_i$, $t \in \{0,\ldots,T-\rho_{ij}\}$. La contrainte (3.18) garantit qu'une machine ne peut exécuter plusieurs tâches en même temps. La contrainte (3.19) concerne la gestion des ressources et intègre correctement le fait que plusieurs ressources peuvent être utilisées en parallèle. En effet, si $u_{ijt} = 1$, c'est-à-dire si la tâche J_i démarre son exécution à la date t sur la machine M_j, alors les $\Gamma_{\varphi_i} - 1$ ressources restantes sont disponibles pour l'exécution d'une autre tâche nécessitant la même ressource, sur une autre machine.

Notons que dans le cas où toutes les ressources sont en unique exemplaire, on a $\Gamma_{\varphi_i} = 1$ ce qui implique que $\Gamma_{\varphi_i} - 1 = 0$, et le membre de gauche de l'inégalité devient $M'(1-u_{ijt})$, et avec M' suffisamment grand, on peut donc réunir les deux inégalités ci-dessus en une seule inégalité, comme cela a déjà été présenté.

De plus, si on note $\mathcal{G}_i$ l'ensemble des indices des machines sur lesquelles aucun des exemplaires de la ressource requise par J_i n'est présent initialement, la contrainte (3.5) devient :

$$\sum_{i=1}^{n} \sum_{j \in \mathcal{G}_i} u_{ij0} = 0.$$

3. Si on veut intégrer les deux aspects précédents, il suffit de remplacer la contrainte (3.7) par :

$$M'(1-u_{ijt}) \geq \sum_{\substack{i_0 | j \in \mathcal{M}_{i_0} \\ i_0 \neq i}} \sum_{t_0=t}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{iji_0})} u_{i_0 j t_0} \tag{3.20}$$

et

$$M'(1-u_{ijt})+(\Gamma_{\varphi_i}-1)\geq \sum_{\substack{i_0|\varphi_{i_0}=\varphi_i\\ i_0\neq i}}\sum_{\substack{j_0\in\mathcal{M}_{i_0}\\ j_0\neq j}}\sum_{t_0=t}^{\min(T-\rho_{i_0j_0},t+\rho_{ij}-1+T_{jj_0})} u_{i_0j_0t_0} \quad (3.21)$$

pour tous $i\in\{1,\ldots,n\}$, $j\in\mathcal{M}_i$ et $t\in\{0,\ldots,T-\rho_{ij}\}$.

3.2.4 Renforcement des Programmes Linéaires en Nombres Entiers

La contrainte (3.15) assure que si une tâche J_i commence son exécution sur une machine M_j au temps t, aucune tâche J_{i_0} $(i_0\neq i)$ qui requiert la même ressource auxiliaire que i ne peut commencer son exécution sur une autre machine que M_j entre t et $t+\rho_{ij}$. La contrainte (3.14) assure que si une tâche J_i commence son exécution sur une machine M_j au temps t, alors aucune tâche J_{i_0} $(i_0\neq i)$ ne pourra commencer sur M_j entre t et $t+\rho_{ij}-1+\beta_{iji_0}$.

Il est possible alors de renforcer ces contraintes. La contrainte (3.14) devient :

$$u_{ijt}+\sum_{j_0\in\mathcal{M}_{i_0}\setminus\{j\}}\sum_{t_0=\max(0,t-\rho_{i_0j_0})}^{\min(T-\rho_{i_0j_0},t+\rho_{ij})}u_{i_0j_0t_0}+\sum_{j_0\in\mathcal{M}_{i_0}\cap\{j\}}\sum_{t_0=\max(0,t-\rho_{i_0j_0}+1)}^{\min(T-\rho_{i_0j_0},t+\rho_{ij}-1)}u_{i_0j_0t_0}\leq 1 \quad (3.22)$$

En effet, si une tâche J_i commence son exécution sur une machine M_j au temps t, alors aucune tâche J_{i_0} $(i_0\neq i)$, qui requiert la même ressource auxiliaire que i, ne peut commencer son exécution sur une autre machine M_{j_1} entre $t-\rho_{i_1j_1}$ et t.

La contrainte (3.15) devient quant à elle :

$$u_{ijt}+\sum_{t_0=\max(0,t-\rho_{i_0j}+1-\beta_{i_0ji})}^{\min(T-\rho_{i_0j},t+\rho_{ij}-1+\beta_{iji_0})}u_{i_0jt_0}\leq 1 \quad (3.23)$$

En effet, si une tâche J_i commence son exécution sur une machine M_j au temps t, alors aucune tâche J_{i_0} $(i_0\neq i)$ ne peut commencer sur M_j entre $t-\rho_{i_0j}+1-\beta_{i_0ji}$ et t.

3.2.5 Inégalités valides

Coupes sur la capacité des machines

Des PLNE de problèmes d'ordonnancement avec indices de temps ont été proposés dans le cas à une machine (voir par exemple [19], [2] et [34]). Les contraintes

classiques modélisant le fait qu'une machine ne peut exécuter qu'une tâche à la fois se généralisent facilement au cas de plusieurs machines (voir par exemple [34] et [70]). Cependant, cette adaptation s'avère plus ardue lorsque des temps de setup dépendant de la séquence sont considérés. C'est principalement la raison pour laquelle les contraintes du problème que l'on traite ne peuvent être formulées uniquement avec ces contraintes. On peut néanmoins les rajouter en vue de les utiliser comme des inégalités valides, permettant éventuellement d'obtenir une meilleure borne de relaxation en programmation linéaire, et couper des solutions fractionnaires.

Ces contraintes s'écrivent comme suit :

$$\sum_{i \in Q_j} \sum_{t_0=\max(0,t-\rho_{ij}+1)}^{\min(T-\rho_{ij},t)} u_{ijt_0} \leq 1, \tag{3.24}$$

pour tout $j \in \{1, \ldots, m\}$ et $t \in \{0, \ldots, T - \min_{i \in Q_j}\{\rho_{ij}\}\}$.

Ces inégalités expriment le fait qu'une machine ne peut exécuter une tâche pour laquelle elle est qualifiée que si aucune autre tâche n'est planifiée sur cette machine au même moment.

Coupes sur la gestion des ressources auxiliaires

On peut s'inspirer des inégalités (3.24) pour exprimer la non-simultanéité d'exécution des tâches qui nécessitent la même ressource auxiliaire. Ces contraintes peuvent s'écrire de la manière suivante :

$$\sum_{i|\varphi_i=k} \sum_{j \in \mathcal{M}_i} \sum_{t_0=\max(0,t-\rho_{ij}+1)}^{\min(T-\rho_{ij},t)} u_{ijt_0} \leq 1, \tag{3.25}$$

pour tout $k \in \{1, \ldots, \ell\}$ et $t \in \{0, \ldots, T\}$.

Cependant, ces contraintes ne permettent pas d'intégrer le temps de transfert des ressources auxiliaires. De façon analogue, les contraintes classiques exprimant le fait qu'une machine ne peut exécuter plus d'une tâche simultanément, ne suffisent pas en présence de temps de setup dépendant de la séquence.

Rajoutées comme inégalités valides à la formulation PLNE du problème, ces contraintes améliorent encore la borne obtenue par relaxation linéaire.

Coupes sur le déplacement des ressources auxiliaires

On peut ajouter des coupes pour intégrer le temps de déplacement unitaire des ressources auxiliaires, comme suit :

$$\sum_{t_0=t}^{\min(T-\rho_{ij},t+\rho_{ij}-1)} u_{ijt_0} + \sum_{\substack{i_0|\varphi_{i_0}=\varphi_i \\ i_0\neq i}} \sum_{\substack{j_0\in\mathcal{M}_{i_0} \\ j_0\neq j, t+\rho_{ij}\leq T-\rho_{i_0j_0}}} u_{i_0j_0(t+\rho_{ij})} \leq 1, \tag{3.26}$$

pour tout $i \in \{1,\ldots,n\}$, $j \in \mathcal{M}_i$ et $t \in \{0,\ldots,T-\rho_{ij}\}$.

Ces contraintes permettent de gérer le temps de déplacement des ressources auxiliaires dans le cas où ce temps est constant et égal à 1. Si une tâche J_i commence entre l'instant t et l'instant $t+\rho_{ij}-1$ sur une machine M_j qualifiée, alors aucune autre tâche J_{i_0} nécessitant la même ressource auxiliaire ne peut commencer sur une machine à l'instant $t+\rho_{ij}$. En effet, la tâche J_i doit d'abord terminer son exécution et il faut un temps de déplacement de ressource auxiliaire. Les contraintes (3.26) permettent de couper des solutions fractionnaires que les inégalités (3.24) et (3.25) ne séparent pas.

3.2.6 Heuristique primale

Les PLNE (FIT-1) et (FIT-2) se résolvent principalement à l'aide de solveurs standards qui reposent sur la combinaison de méthodes par séparation et évaluation, dites de *Branch & Bound* et de génération de coupes, dite de *Branch & Cut*. Une relaxation continue du problème est résolue à l'optimum par programmation linéaire (la méhode utilisée étant souvent l'algorithme du simplexe de Dantzig [68], d'autres méthodes sont aussi applicables). Puis le problème est scindé en deux sous-problèmes distincts. Cette séparation est faite sur une variable. Dans l'un des sous-problèmes, elle est fixée à 0 et dans l'autre elle est fixée à 1. Comme c'est une variable binaire, aucune solution du problème n'est perdue dans cette séparation. Le même procédé se répète de façon arborescente, et lorsqu'une solution réalisable (donc entière) est trouvée, une borne du coût de la solution optimale est mise à jour. Les solutions optimales des sous-problèmes relâchés (en variables réelles) étant meilleures que les solutions entières, une autre borne est maintenue, de sorte à permettre, soit d'éviter l'exploration dans un sous-problème (car la meilleure solution entière trouvée a un coût meilleur que la borne de relaxation en ce sous-problème), soit de terminer la résolution (si c'est le cas de tous les sous-problèmes ouverts).

Une technique qu'il convient d'intégrer à cet algorithme est l'ajout d'une procédure qui, étant donné un sous-problème (correspondant donc à un nœud de l'arbre de résolution) avec un certain nombre de variables fixées, construit une solution entière réalisable du problème. Cela a pour but de mettre à jour de façon régulière la borne (qui est un majorant quand il s'agit d'une minimisation et un minorant quand il s'agit d'une maximisation), appelée aussi *borne primale*.

C'est le principe de *l'heuristique primale*. Dans notre problème, une heuristique primale a été implémentée dans ce but. Il s'agit d'un algorithme qui, étant donnée une solution $(u^*_{ijt})_{i\in\{1,\dots,n\},j\in\mathcal{M}_i,t\in\{0,\dots,T-\rho_{ij}\}}$ fractionnaire, avec un sous-ensemble F^* de variables fixées, retourne une solution $(\tilde{u}_{ijt})_{i\in\{1,\dots,n\},j\in\mathcal{M}_i,t\in\{0,\dots,T-\rho_{ij}\}}$ réalisable de variables binaires.

```
1  ORDRE-TÂCHES((u*_ijt)_{i∈{1,...,n},j∈M_i,t∈{0,...,T−ρ_ij}}, F*))
2  k ← 1
3  pour i ← 1 à n faire
4      estRange[i] ← 0
5  pour i ← 1 à n faire
6      pour j ∈ M_i faire
7          pour t ← 0 à T − ρ_ij faire
8              si (i, j, t) ∈ F* alors
9                  si estRange[i] = 0 alors
10                     ordre[k] ← i
11                     estRange[i] ← 1
12                     k ← k + 1
13 pour i ← 1 à n faire
14     si estRange[i] = 0 alors
15         ordre[k] ← i
16         estRange[i] ← 1
17         k ← k + 1
18 retourner ordre
```

Algorithme 3: Ordre d'instanciation des tâches dans l'heuristique primale

L'algorithme 4 décrit l'heuristique primale. L'algorithme 3 construit l'ordre de parcours des tâches afin de les instancier à des valeurs binaires. Cet ordre est choisi de sorte à privilégier les tâches J_i pour lesquelles il existe une variable u_{ijt} fixée dans le nœud courant. En effet, F^* désigne ici l'ensemble des triplets (i, j, t) pour lesquelles la variable u_{ijt} est fixée. Pour les variables fixées, l'ordre suit l'indice des

tâches. De même pour les tâches non concernées par les variables fixées.

```
1  PRIMAL((u*_ijt)_{i∈{1,...,n},j∈M_i,t∈{0,...,T−ρ_ij}}, F*)
2  ordre ←ORDRE-TÂCHES((u*_ijt)_{i∈{1,...,n},j∈M_i,t∈{0,...,T−ρ_ij}}, F*)
3  pour i ← 1 à n faire
4      pour j ∈ M_i faire
5          pour t ← 0 à T − ρ_ij faire
6              ũ_ijt ← −1
7  pour ind ← 1 à n faire
8      i ← ordre[ind]
9      indj ← 1 reste ← true
10     tant que indj ≤ |M_i| et reste = vrai faire
11         j ← M_i[indj]
12         t ← 0
13         tant que t ≤ T − ρ_ij et (ũ_ijt = 0 ou ((i, j, t) ∈ F* et u*_ijt = 0) ou
           (t = 0 et R_{φ_i} ≠ j)) faire
14             t ← t + 1
15         si t ≤ T − ρ_ij alors
16             reste ← faux
17         indj ← indj + 1
18     si reste = true alors
19         retourner « Erreur : Pas de solution réalisable possible »
20     maxj ← M_i[indj − 1], maxt ← t
21     pour indj_0 ← indj à |M_i| faire
22         j_0 ← M_i[indj_0]
23         pour t_0 ← t à T − ρ_{i_0 j_0} faire
24             si ((t_0 ≠ 0) ou (R_{φ_i} = j_0)) et (u*_{ij_0t_0} > u*_{i(maxj)(maxt)} et ũ_ijt ≠ 0)
               alors
25                 maxj ← j_0, maxt ← t_0
26     pour j ∈ M_i faire
27         pour t ← 0 à T − ρ_ij faire
28             si maxj = j et maxt = t alors
29                 ũ_ijt ← 1
30             sinon
31                 ũ_ijt ← 0
32     pour i_0 ≠ i tel que maxj ∈ M_{i_0} faire
33         pour t_0 ← max(0, maxt − ρ_{i_0(maxj)} + 1 − β_{i_0(maxj)i}) à
           min(T − ρ_{i_0(maxj)}, maxt + ρ_{i(maxj)} − 1 + β_{i(maxj)i_0}) faire
34             si ũ_{ij_0t_0} = 1 alors
35                 retourner « Erreur : Une contrainte violée »
36             ũ_{ij_0t_0} ← 0
37     pour i_0 ≠ i tel que φ_i = φ_{i_0} faire
38         pour j_0 ∈ M_{i_0} \ {maxj} faire
39             pour t_0 ← max(0, maxt − ρ_{i_0j_0}) à min(T − ρ_{i_0j_0}, maxt + ρ_{i(maxj)})
               faire
40                 si ũ_{i_0j_0t_0} = 1 alors
41                     retourner « Erreur : Une contrainte violée »
42                 ũ_{i_0j_0t_0} ← 0
43 retourner (ũ_ijt)_{i∈{1,...,n},j∈M_i,t∈{0,...,T−ρ_ij}}
```

Algorithme 4: Heuristique primale

Dans l'algorithme 4, la solution entière est construite, à partir de l'ordre déterminé à la ligne 2. Pour toute tâche J_i, un couple (j,t) est déterminé afin d'instancier la variable $\tilde{u}_{ijt}$ à 1 (ligne 29). Ce couple est choisi tel que u_{ijt} n'est pas fixée à 0 dans le nœud courant et tel que l'instanciation de cette variable à 1 ne viole aucune contrainte du problème ; parmi toutes les variables satisfaisant cette propriété, celle dont la valeur est maximale dans la solution optimale fractionnaire est retenue. Les lignes 9 à 25 décrivent le calcul de ce couple (j,t). Ce choix fait de l'heuristique une *méthode d'arrondi*.

Des lignes 26 à 31, la variable correspondant au couple choisi est instanciée à 1 dans la solution entière primale et les autres variables $u_{ij't'}$ sont nulles dans cette solution. Les lignes 32 à 42 garantissent que les contraintes 3.14 et 3.15 sont satisfaites. En effet, quand une variable est instanciée à 1, ces contraintes imposent que d'autres variables soient nulles. Cela est utilisé dans les itérations suivantes pour déterminer quelles variables ne peuvent plus être égales à 1.

Remarque 5. La ligne 19 de l'algorithme 4 se réfère au cas où aucune solution primale réalisable n'a pu être construite car aucun couple (j,t), pour une tâche J_i donnée, n'a pu être choisi. Cela est possible si les seules variables susceptibles d'être choisies sont instanciées à 0.

Les lignes 35 et 41 correspondent au cas où l'une des contraintes du programme linéaire est violée. Dans ce cas, l'algorithme retourne une erreur et aucune solution n'est obtenue.

Proposition 11. *Si l'algorithme 4 se termine sans erreur, alors la solution retournée est réalisable.*

Démonstration. À chaque itération de la boucle *pour* de la ligne 7, une et une seule variable est instanciée à 1, ce qui permet de vérifier la satisfaction des contraintes (3.2). Par ailleurs, à chaque fois que l'une des contraintes (3.14) ou (3.15) est violée, l'algorithme s'arrête avec une erreur aux lignes 35 et 41.

Enfin, la satisfaction de la contrainte (3.16) est garantie par les tests des lignes 13 et 24. En effet, une variable u_{ijt} telle que $t = 0$ et $R_{\varphi_i} \neq j$ ne pourra être choisie pour être instanciée à 1.

Ainsi, si l'algorithme ne retourne pas d'erreur, la solution obtenue vérifie nécessairement toutes les contraintes de la formulation (FIT-2). Par définition, elle est donc réalisable. □

L'apport de cette méthode d'arrondi à la qualité de la résolution a été analysée et les résultats sont présentés en section 3.4.1 sur des instances de petite taille.

3.3 Minimiser le nombre de transferts

Les résultats de la section 2.4 nous amènent à considérer la Programmation Linéaire en Nombre Entiers (PLNE), en particulier une formulation existante pour le problème de couverture minimale. L'énoncé suivant fournit alors une méthode de résolution exacte du problème $\Pi'_{J,M,A}$ au moyen de cette technique d'optimisation.

La formulation suivante est une formulation valide pour le problème $\Pi'_{J,M,A}$.

$$\min \sum_{i=1}^{\ell} \sum_{j=0}^{m} u_{ij} \tag{3.27}$$

s. c.

$$\sum_{j|k\in M_j(i)} u_{ij} \geq 1 \qquad \forall i = 1,\dots,\ell, \quad \forall J_k \in E_i \cup \{J_0\} \tag{3.28}$$

$$u_{iR_i} = 1 \qquad \forall i = 1,\dots,\ell \tag{3.29}$$

$$u_{ij} \in \{0,1\} \qquad \forall i = 1,\dots,\ell, \quad \forall j = 0,\dots,m \tag{3.30}$$

On considère, pour tout $i \in \{1,\dots,\ell\}$, $j \in \{0,\dots,m\}$, que la variable $u_{ij} \in \{0,1\}$ est égale à 1 si et seulement si la ressource auxiliaire A_i passe sur la machine M_j dans la solution.

Ainsi, on voit que la valeur $\sum_{j=0}^{m} u_{ij}$ est le nombre total de machines qui utilisent A_i dans une solution. Soit alors

$$(x,y) = (x^1,\dots,x^\ell,y^1,\dots,y^\ell)$$

la solution correspondante. On a

$$f_i(x^i,y^i) + 1 = \sum_{j=0}^{m} u_{ij}.$$

Cela implique que

$$\sum_{i=1}^{\ell} \left(\sum_{j=0}^{m} u_{ij} \right) = \sum_{i=1}^{\ell} \big(f_i(x^i,y^i) + 1 \big) = \tilde{f}(x,y) + \ell.$$

Ainsi, la fonction objectif du programme linéaire correspond bien à la fonction objectif du problème $\Pi'_{J,M,A}$.

En guise de contrainte du problème, il est imposé que pour toute ressource auxiliaire A_i et chacune des tâches nécessitant cette ressource, il existe au moins une machine qualifiée pour cette tâche par laquelle passe A_i. En effet, si cette contrainte n'est pas vérifiée, il est impossible d'exécuter toutes les tâches.

Les contraintes (3.28) permettent de vérifier ce fait car l'ensemble

$$D_k = \{j \mid k \in M_j(\varphi_k)\}$$

est l'ensemble des indices des machines pouvant exécuter la tâche J_k.

Nous avons de plus, par les contraintes (3.29), vérifié le fait qu'une ressource auxiliaire passe par la machine à laquelle elle est initialement affectée. Toute solution réalisable de ce PLNE est donc valide pour $\Pi'_{J,M,A}$ et réciproquement.

Ainsi, à partir d'une solution de $\Pi'_{J,M,A}$, on peut construire, d'après la proposition 4 du chapitre 2, une solution pour $\Pi_{J,M,A}$ en complétant celle obtenue par le PLNE.

3.4 Résultats expérimentaux

Dans ce qui suit, des tests numériques sont présentés et permettent d'analyser les performances des trois modèles en programmation linéaire en nombres entiers. La formulation à variables indexées par le temps a été testée avec les contraintes de type *big-M* et comparée au modèle dans lequel elles sont remplacées par les contraintes avec membre de droite égal à 1, données par les inégalités (3.14) et (3.15). Ces exécutions ont été lancées sur des instances du problème générées aléatoirement de sorte à faire ressortir les principales caractéristiques des données industrielles et à garder un certain degré de généralité, et ce en vue de conserver toute la difficulté du problème. Ces expérimentations ont été effectuées sur un processeur Intel Core i7-2640M CPU, 2.80GHz avec le solveur IBM ILOG-CPLEX, version 12.5.

Dans ces instances créées aléatoirement, les tâches ont en général un nombre de plaquettes de 25 mais avec une probabilité de 10%, ce nombre est compris entre 1 et 25. Le nombre maximal de machines qualifiées pour une tâche est fixé à $\max(2, \lfloor \frac{m}{2} \rfloor$. Les durées d'exécution sont dans $\{1, \ldots, 10\}$ et les durées de setup dans $\{0, \ldots, 5\}$. Enfin, au plus 50% des paires de tâches qui peuvent être traitées par une machine commune ont un temps de setup non nul.

Les tables 3.1-3.4 présentent les résultats obtenus sur les instances, qui sont décrites au format $(N - M - A)$, excepté pour le problème de maximisation du nombre de plaquettes dans une fenêtre de temps, pour lequel il y a un quatrième paramètre (l'horizon H).

3.4.1 Formulations à variables indexées par le temps

La table 3.1 présente l'impact de l'heuristique primale présentée à la section 3.2.6 sur la résolution du programme linéaire à variables indexées par le temps

Inst.	Sans Heuristique primale			Avec Heuristique primale		
	Durée (s)	# Nœuds	# Act. B	Durée (s)	# Nœuds	# Act. B
(8-2-3) (1)	1,32	1	1	1,31	1	3
(8-2-3) (2)	1,05	1	1	1,07	1	3
(8-2-3) (3)	4,09	1	1	2,64	1	3
(8-2-3) (4)	2,23	1	1	1,69	1	4
(8-2-3) (5)	7,37	1	1	6,70	1	6
(10-2-3) (1)	22,06	1	1	21,64	1	2
(10-2-3) (2)	19,16	1	2	24,08	1	5
(10-2-3) (3)	15,14	1	1	29,10	1	9
(10-2-3) (4)	15,96	1	2	14,94	1	8
(10-2-3) (5)	17,97	1	1	18,21	1	5
(15-2-3) (1)	198,00	1	1	291,96	1	12
(15-2-3) (2)	312,07	41	2	**258,86**	**16**	**8**
(15-2-3) (3)	777,74	170	2	**526,31**	**165**	**7**
(15-2-3) (4)	1427,55	1927	7	**1009,27**	**977**	**9**
(15-2-3) (5)	1144,08	899	5	**1131,91**	**628**	**10**

TABLE 3.1 – Résultats obtenus pour $\sum_j w_j C_j$ (Π_2) avec la formulation PLNE à variables indexées par le temps, avec et sans heuristique primale d'arrondi.

(FIT-1). Deux colonnes affichent les performances de la résolution avec et sans l'utilisation de cette heuristique d'arrondi. La durée de résolution, en secondes, le nombre de nœuds explorés dans l'arbre de recherche, ainsi que le nombre de fois que la borne supérieure primale est actualisée, sont indiqués dans cette table. Ces expérimentations ont été effectuées sur cinq instances à 8 tâches, cinq instances à 10 tâches et cinq instances à 15 tâches, toutes comprenant 2 machines et 3 masques.

Dans tous les cas, la borne supérieure est améliorée beaucoup plus souvent en présence de l'heuristique primale. Celà prouve qu'elle intervient très souvent dans la mise à jour de la borne primale. Sur les instances à 8 tâches, on observe une légère amélioration du temps de calcul, mais l'impact de l'heuristique primale commence à devenir évident sur les instances à 15 tâches, où par exemple sur l'instance 4, on arrive à gagner 400 secondes de résolution et jusqu'à 1000 nœuds explorés en moins.

L'augmentation du temps de calcul, notamment observé sur certaines instances

à 10 tâches, indique que la contrepartie de l'ajout d'une heuristique primale peut se ressentir sur la durée de résolution. Ce temps supplémentaire se compense largement sur de plus grandes instances puisqu'il permet d'accélérer la réduction de l'écart d'optimalité (*gap*) potentiel (écart entre bornes inférieure et supérieure). À titre d'exemple, notons que pour l'instance 5 à 15 tâches, un écart d'optimalité potentiel inférieur à 10% est atteint en 417 secondes alors qu'il faut attendre 715 secondes sans l'heuristique primale. Il est atteint en 396 secondes sur l'instance 4 alors qu'il faut 667 secondes sans l'heuristique primale.

Inst.	Sans inég. valides		Avec contraintes (3.24)		Meilleure solution
	big-M	RHS 1	big-M	RHS 1	connue
10-2-4	69.9	95.9	137.3	143.21	**170**
10-3-3	73.02	95.42	107.04	113.64	**134**
15-2-3	88.81	132.9	210.09	216.75	**278**
15-3-10	100.33	125.95	195.58	200.39	**246**
15-3-3	86.27	120.94	148.35	160.66	**214**
15-3-5	94.31	137.52	168.53	182.11	**229**
16-3-6	100.83	137.78	198.08	206.35	**243**
20-2-3	106.66	167.24	322	329.94	**485**
20-3-3	146.99	206.92	336.44	348.63	**478**
25-2-3	172.90	257	686.57	690.94	**872**
25-3-3	152.41	220.45	387.37	396.15	547
25-3-12	156.78	216.21	420.56	431.16	531
30-3-4	192.62	280.14	577.30	589.91	**756**
32-4-8	230.57	311.54	523.40	542.59	723
40-3-4	270.47	393.44	1004.02	1014.22	1291
50-3-6	321.22	456.21	1448.01	1456.81	1781

TABLE 3.2 – Améliorations de la borne inférieure, apportées par les inégalités valides pour la minimisation de $\sum_j w_j C_j$ avec la formulation à variables indexées par le temps.

Les expérimentations menées concernant la minimisation du critère $\sum_j w_j C_j$ et la maximisation du nombre de plaquettes dans une fenêtre de temps montrent que la formulation retenue ne fournit des solutions optimales en temps raisonnable que sur des instances de taille limitée. Toutes les instances à 10 tâches et 2 machines qui ont été retenues sont résolues à l'optimum. Celles à 20 tâches, ou encore à

Inst.	Π_2 Durée		**Inst.**	Π_1 Durée	
	big-M	RHS 1		big-M	RHS 1
(8-2-3)	0.90s	1.60s	**(8-2-3-20)**	5.82s	3.92s
(10-2-3)	8.82s	9.89s	**(10-2-3-25)**	38.14s	44.94s
(10-3-3)	1.84s	2.50s	**(10-3-3-16)**	3.37s	2.59s
(15-2-3)	126.84s	188.87s	**(15-2-3-37)**	5h33mn	6h20mn
(15-3-3)	22.19s	19.98s	**(15-3-3-25)**	8m10s	7m30s
(20-2-3)	3h57mn	2h45mn	**(20-2-3-50)**	$>$ 3 jours	$>$ 3 jours
(20-3-3)	1h58mn	1h59mn	**(20-3-3-33)**	$>$ 3 jours	$>$ 3 jours

TABLE 3.3 – Résultats obtenus pour $\sum_j w_j C_j$ (Π_2) et la maximisation du nombre de plaquettes (Π_1) avec la formulation PLNE à variables indexées par le temps.

15 tâches et 2 machines ne peuvent être résolues en moins d'une heure. Avec 20 tâches et 3 machines, seule une instance est résolue à l'optimum. La formulation avec contraintes de *big-M* est beaucoup moins efficace sur les instances à 20 tâches et cela est dû au fait qu'elle fournit de mauvaises bornes en relaxation continue. De plus, on obtient généralement de bien meilleurs résultats avec Π_2 qu'avec Π_1.

Les inégalités valides ajoutées à la formulation fournissent de meilleures bornes. La table 3.2 montre les améliorations de cette borne en relaxation continue. On peut observer celles apportées par les contraintes de type *big-M* (colonne *big-M*) et celle obtenues avec les contraintes de membre de droite égal à 1 (colonne *RHS 1*). Les valeurs en gras dans la colonne *Meilleure solution connue* indiquent que la solution est garantie optimale. Les contraintes (3.24) améliorent sensiblement cette borne. Sur des instances de grande taille, la différence est plus significative. Dans certains cas, elle est cinq fois meilleure avec les inégalités valides. Elles peuvent alors s'avérer très pertinentes pour résoudre de plus grandes instances à l'optimalité. L'amélioration en question est très légère lorsqu'on remplace les contraintes de type *big-M* par les contraintes (3.14) et (3.15). Ainsi, les contraintes (3.24) resserrent l'écart entre les deux modèles.

3.4.2 Minimisation du nombre de transferts de ressources auxiliaires

L'approche basée sur le problème de couverture minimale d'ensembles (*Set Cover*) fonctionne très bien sur des instances réelles. Comme ici le nombre de ma-

Inst.	Transferts de ressources	
	Gap(%)	Durée (s)
(8-2-3)	0	1
(32-4-8)	0	1
(128-15-15)	0	1
(500-18-25)	0	1
(1600-20-25)	0	1

TABLE 3.4 – Résultats pour le problème de minimisation du nombre de transferts de ressources avec la formulation basée sur le problème de couverture (Set Cover).

chines reste peu élevé, le nombre d'ensembles couvrants est très faible, et les durées de résolution sont ainsi exceptionnellement faibles avec cette méthode. La table 3.4 donne une idée de cette efficacité. En effet, on observe que jusqu'à 1600 tâches, la solution optimale est obtenue (la colonne *Gap* indique que l'écart d'optimalité est nul, donc que la solution retournée est bien optimale). Le temps de résolution est toujours de l'ordre de la seconde (voire inférieur). Cela se comprend, au vu de la complexité du problème, analysée au chapitre 2 qui dépend surtout du nombre de machines.

3.5 Conclusion

Dans ce chapitre, une formulation en programmation linéaire en nombres entiers (PLNE) des problèmes de minimisation de $\sum_j w_j C_j$ et de maximisation de $\sum_j c_j \theta_j$ est proposée. On y trouve également une réécriture de la formulation pour la minimisation du nombre de transferts de ressources auxiliaires. Contrairement à cette dernière formulation, le PLNE proposé pour les deux autres problèmes contient un nombre de variables qui dépend de l'horizon de temps considéré T. Le problème associé est que T peut prendre une valeur très élevée si les temps d'exécution et les temps de setup sont importants. Des inégalités valides viennent renforcer la formulation PLNE et permettent d'améliorer les bornes de relaxation linéaire.

De plus, une heuristique primale a été incorporée à l'algorithme de résolution afin d'accélérer l'obtention de solutions réalisables. Des expérimentations ont été menées afin d'évaluer la performance de ces formulations à l'aide d'un logiciel commercial. Par ailleurs, dans le cadre de l'étude de ce problème d'ordonnancement sous l'aspect *multi-critère*, développée au chapitre 5, nous utiliserons ce modèle

dans une méthode basée sur des fonctions d'agrégation (ou fonctions *scalarisantes*) qu'on peut exprimer linéairement en fonction des valeurs des critères.

Chapitre 4

Résolution approchée du problème monocritère

Les résultats acquis dans les chapitres précédents, notamment concernant la complexité théorique des problèmes étudiés, amènent à considérer des méthodes de résolution adaptées pour des instances de taille industrielle. Dans ce qui suit, des approches de résolution non exacte, avec et sans garantie de performance, sont proposées pour les trois problèmes traités. Un algorithme métaheuristique est détaillé pour l'optimisation du nombre de plaquettes traitées dans une fenêtre de temps fixée et la somme pondérée des dates de fin d'exécution. Dans la suite du chapitre, la minimisation du nombre de transferts de ressources auxiliaires est abordée sous un angle plus théorique et résolue avec un algorithme d'approximation inspiré d'une méthode classique. Les résultats apportés par ces approches sont présentés et analysés en fin de chapitre.

4.1 Algorithme mémétique

4.1.1 Introduction

En guise de préambule, décrivons brièvement le principe d'un algorithme mémétique. L'idée de base d'un algorithme mémétique est la combinaison de deux principes d'optimisation approchée, les algorithmes génétiques et la recherche locale. Ces méthodes couvrent des aspects distincts de la recherche d'une solution lorsque la combinatoire est élevée.

Les algorithmes génétiques jouent un rôle dans la *diversification*, permettant de générer des solutions distinctes et relativement éloignées les unes des autres. Cette notion de distance entre solutions dépend du problème traité et traduit le souci de couvrir de façon efficace l'ensemble des solutions. Il s'agit d'*algorithmes évolutionnaires* (algorithmes s'inspirant de la théorie de l'évolution pour manipuler des solutions dans le but de les faire évoluer et idéalement les faire converger vers des solutions optimales), qui utilisent itérativement des tirages aléatoires ; on peut en ce sens les considérer comme des algorithmes stochastiques. Ce type de

méthodes a été pour la première fois utilisé par John Holland [36] [73] en 1975. Ce concept a depuis bien évolué. Ils manipulent des *populations* de solutions, c'est-à-dire un ensemble de solutions qui seront itérativement *sélectionnées*, *croisées* et *mutées* pour les améliorer au sens d'un critère donné. Un *critère d'arrêt*, condition devant être vérifiée pour que l'exécution de l'algorithme s'arrête (correspondant à l'arrêt de l'exécution de la boucle principale), doit être défini et il existe plusieurs manières de le faire. Il y a diverses stratégies pour effectuer ces opérations ainsi que pour le choix des solutions de la population de l'itération suivante. Ces stratégies fondent la spécificité de chaque algorithme.

L'opération de mutation est celle qui utilise la recherche locale [66]. Les algorithmes de recherche locale interviennent pour, étant donné une solution initiale de l'espace de recherche, la modifier et atteindre la meilleure solution possible parmi celles qui en sont proches. À nouveau, la notion de proximité ici est définie à partir du problème traité et de la structure des solutions, ainsi que des mouvements permettant de transformer une solution en une autre. Cette étape est la phase d'*intensification*. Elle consiste donc, à chaque itération, à passer d'une solution à une autre, par une transformation appelée *opération de voisinage*. La solution obtenue est dite *solution voisine*. La stratégie utilisée, aussi bien pour les opérations de voisinage que pour la gestion de la solution conservée à chaque itération, varie d'un problème à l'autre. C'est le choix de la structure de la solution ainsi que des opérateurs de voisinage qui s'y appliquent, qui fonde la particularité de chaque algorithme. Le schéma générique de diversification est ici complété par une méthode d'intensification adaptée au problème traité. L'ensemble forme l'algorithme mémétique.

Les algorithmes mémétiques sont des métaheuristiques souvent utilisées pour la résolution de problèmes d'optimisation combinatoire. Ils permettent une résolution générique, dans laquelle les spécificités du problème se retrouvent dans la recherche locale, notamment dans la notion de voisinage et de distance entre solutions. Leur efficacité est reconnue (voir par exemple les résultats obtenus pour le problème de coloration de graphes [64] qui sont les meilleurs connus, le problème de partition de graphe [59], de stable maximal [21], et même d'ordonnancement sur machines parallèles [61]) et ils apportent bien souvent de meilleures solutions que d'autres méthodes de résolution non exacte, sur des problèmes pour lesquels il n'existe pas d'algorithme polynomial connu.

Un autre avantage de cette méthode est de pouvoir analyser le compromis entre les phases d'intensification et de diversification à travers les nombreux paramètres de l'algorithme, notamment le critère d'arrêt utilisé pour l'une et l'autre de ces étapes. Ces éléments jouent un rôle déterminant dans la qualité des solutions obtenues, selon la taille et la structure des instances traitées. Nous verrons cet effet à la section 4.1.9.

Dans notre cas, les résultats de complexité fournis au chapitre 2 justifient le recours à cette classe de méthodes. Il est présenté dans ce qui suit l'algorithme mémétique utilisé pour résoudre la maximisation du nombre de plaquettes produites dans un horizon de temps fixé et la minimisation de la somme pondérée des dates de fin d'exécution des tâches. Dans une première partie, la représentation des individus est décrite avec précision. Puis, le fonctionnement général de l'algorithme est présenté, avec un pseudo-code. Les parties qui suivent décrivent en détail tous les aspects de la méthode. Il s'agit en particulier de

- L'initialisation des individus de la population,
- la méthode de sélection,
- la procédure de croisement,
- la mutation, sous forme de recherche locale, avec la description des voisinages utilisés.

Certaines propriétés de cette méthode permettent de la valoriser. On montre alors qu'elle est pertinente en détaillant les caractéristiques des solutions retournées. Puis certaines améliorations notables sont proposées en fin de chapitre et accompagnent des tests comparatifs effectués sur des instances générées aléatoirement, mais sur le modèle des instances d'industrie.

Dans la suite, on utilise les notations suivantes, relatives aux données du problème. Ces notations sont par ailleurs utilisées dans tout le document :

- L'ensemble de n tâches $J = \{J_1, \ldots, J_n\}$.
- L'ensemble de m machines $M = \{M_1, \ldots, M_m\}$.
- L'ensemble de ℓ ressources auxiliaires $A = \{A_1, \ldots, A_\ell\}$, $\ell \leq n$.
- Pour toute tâche J_i, son nombre de *plaquettes* $c_i \in \mathbb{N}$, sa priorité w_i, son indice de ressource auxiliaire requise $\varphi_i \in \{1, \ldots, \ell\}$ et son ensemble d'indices de machines qualifiées $\mathcal{M}_i \subset \{1, \ldots, m\}$.
- Pour toute tâche J_i et toute machine $M_j \in \mathcal{M}_i$ le temps d'exécution $\rho_{ij} \in \mathbb{N}$ de J_i par M_j.
- Pour tout couple de tâches (J_i, J_k) et toute machine M_j, $j \in \mathcal{M}_i \cap \mathcal{M}_k$, le temps de setup $\beta_{ijk} \in \mathbb{N}$ mis par la machine M_j pour exécuter J_k juste après J_i.
- L'horizon de temps $H \in \mathbb{N}$.
- L'emplacement initial $R_i \in \{0, \ldots, m\}$ de chaque ressource A_i.

4.1.2 Codage

```
1  DÉCODER(I, x_1, ..., x_n, y_1, ..., y_n)
2  t_0 ← 0
3  pour j ← 0 à m faire
4      ρ_0j ← 0
5  pour j ← 1 à n faire
6      ρ_j0 ← 0
7  pour i ← 1 à ℓ faire
8      S[i] ← R_i        ▷ Position courante des masques
9      B[i] ← 0          ▷ Dernière tâche exécutée avec chaque masque
10 pour i ← 1 à m faire
11     D[i] ← 0          ▷ Dernière tâche ordonnancée sur chaque machine
12 pour i ← 1 à n faire
13     τ ← 0
14     si S[φ_{x_i}] ≠ y_i alors
15         τ ← 1         ▷ Temps unitaire de transfert
16     t_{x_i} ← max(t_{B[φ_{x_i}]} + ρ_{B[φ_{x_i}]S[φ_{x_i}]} + τ, t_{D[y_i]} + ρ_{D[y_i]y_i} + β_{D[y_i]y_i x_i})
17     D[y_i] ← x_i      ▷ Mise à jour de la dernière tâche exécutée sur
                           la machine
18     B[φ_{x_i}] ← x_i  ▷ Mise à jour de la dernière tâche exécutée
                           avec le masque
19     S[φ_{x_i}] ← y_i  ▷ Mise à jour de la position du masque
20 pour i ← 1 à n faire
21     m_{x_i} ← y_i
22 retourner ((m_1, t_1), ..., (m_n, t_n))
```

Algorithme 5: Algorithme de décodage associé à $f(x_1, \ldots, x_n, y_1, \ldots, y_n)$

Dans un algorithme mémétique, les solutions ne sont en général pas représentées directement comme éléments de l'espace dans lequel elles sont définies. En effet, les solutions sont parfois représentées en mémoire sous forme de structure de données plus ou moins complexes, selon le problème étudié. Maintenir de telles structures dans le cadre d'un algorithme manipulant des populations de solutions s'avère généralement coûteux en espace mémoire, mais aussi en temps de calcul, puisque le maintien de ces structures de données nécessite des opérations supplémentaires. Par exemple, pour un problème d'ordonnancement, la solution doit contenir l'ensemble des tâches planifiées sur chaque machine, mais aussi les dates auxquelles elles débutent. Ces données ne sont pourtant pas toujours nécessaires dans la représentation d'une solution, puisqu'elles se déduisent de l'ordre d'enchaî-

nement des tâches. Les solutions ne sont donc pas toujours représentées comme dans leur ensemble de définition. Il convient de les *coder* en les représentant dans un ensemble (l'espace des individus) dont les éléments seront manipulés dans l'algorithme. À l'issue de la résolution, il faut effectuer un *décodage* pour trouver la solution correspondant à l'individu retourné. Soit Π_1 (resp. Π_2) le problème consistant à maximiser le nombre de plaquettes produites dans un horizon de temps fixé H (resp. minimiser la somme pondérée des dates de fin d'exécution). On note χ l'ensemble des solutions réalisables de Π_1 et Π_2. Dans la suite, on considère une instance quelconque I (resp. I') de Π_1 (resp. Π_2) en utilisant les notations de la section 4.1.1.

Notons $\mathcal{C}$ l'espace des individus, dans lequel les éléments sont utilisés dans l'algorithme. Le codage d'une solution est le suivant : $(x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$ où les x_i sont les indices des tâches ($(x_1, \ldots, x_n)$ représente donc une permutation de l'ensemble des tâches, y_i est l'indice d'une machine (qualifiée pour la tâche J_{x_i})). Ainsi, pour tout élément $(x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$, on a : $\{x_1, \ldots, x_n\} = \{1, \ldots, n\}$ et $y_i \in \mathcal{M}_{x_i}$.

Pour tout individu $(x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$, il existe une unique solution associée $z \in \chi$. On définit donc l'application $f : \mathcal{C} \to \chi$, explicitée par l'algorithme 5, et qui, pour tout $X \in \mathcal{C}$, retourne $f(X) \in \chi$. Toute solution de Π_1 (resp. Π_2) est de la forme $((m_1, t_1), \ldots, (m_n, t_n)) \in \chi$ où pour tout $i \in \{1, \ldots, n\}$, $m_i \in \mathcal{M}_i$ est la machine qui exécute J_i et $t_i \in \mathbb{N}$ sa date de début d'exécution. Dans tous les algorithmes de cette section, on considère que la structure I contient toutes les données du problème.

On voit alors que dans $(x_1, \ldots, x_n, y_1, \ldots, y_n)$, les y_i sont les indices des machines sur lesquelles s'exécutent les tâches d'indices x_i dans la solution donnée par $f(x_1, \ldots, x_n, y_1, \ldots, y_n)$.

Définition 4. Une solution $x = ((m_1, t_1), \ldots, (m_n, t_n)) \in \chi$ est un *ordonnancement au plus tôt* si et seulement si pour tout $i \in \{1, \ldots, n\}$, on ne peut pas diminuer t_i sans modifier les dates de début d'exécution des autres tâches, l'ordre d'exécution des tâches sur la machine M_{m_i}, l'ordre d'exécution des tâches qui requièrent la ressource A_{φ_i}, ni la machine sur laquelle elles sont exécutées, ni leur réalisabilité, c'est-à-dire s'il n'existe pas de solution $((m_1, t_1), \ldots, (m_i, t'_i), \ldots, (m_n, t_n)) \in \chi$ telle que les tâches exécutées sur M_{m_i} le sont dans le même ordre que dans x, les tâches J_k tels que $\varphi_k = \varphi_i$ sont exécutées dans le même ordre que dans x et $t'_i < t_i$.

Proposition 12. *Pour tout élément $(x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$, on a*

$$((m_1, t_1), \ldots, (m_n, t_n)) = f(x_1, \ldots, x_n, y_1, \ldots, y_n) \in \chi$$

et c'est un ordonnancement au plus tôt.

Démonstration. Soit $X = (x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$. Montrons d'abord que l'algorithme DÉCODER permet d'obtenir une solution dans χ. Selon la ligne 21, et la définition de $\mathcal{C}$, la contrainte des machines qualifiées est respectée. Chaque tâche est bien exécutée une et une seule fois.

Les contraintes induites par les temps de setup, la non simultanéité d'exécution de plusieurs tâches par une même machine, la gestion des ressources, sont garanties par l'expression de la ligne 16. Elle exprime le fait que, pour commencer l'exécution d'une tâche sur une machine donnée, deux conditions doivent être réalisées (exprimées par le max) : la mobilisation de la ressource auxiliaire requise (donc la date de fin d'exécution de la dernière tâche exécutée avec cette ressource plus 1 s'il faut la déplacer) et l'écoulement du temps de setup requis depuis la date de fin d'exécution de la dernière tâche exécutée sur cette machine. Ainsi, $f(X) \in \chi$.

Montrons que c'est un ordonnancement au plus tôt. Soit $i \in \{1, \ldots, n\}$; si on remplace la date de début d'exécution de la tâche J_i par $t_i - K$ (K étant un entier strictement positif), en conservant les mêmes valeurs pour les m_j et les $t_j (j \neq i)$, alors au moins l'une des deux conditions précédemment évoquées ne sera plus satisfaite (soit la ressource requise ne sera pas encore disponible, soit la machine sur laquelle la tâche doit être exécutée est occupée). En effet, J_i doit être exécutée après la dernière tâche exécutée sur la machine m_i et après la dernière tâche J_k telle que $\varphi_k = \varphi_i$ qui a été exécutée. L'algorithme permet de l'exécuter au plus tôt après ces deux tâches (en vérifiant les contraintes de ressource et de setup). On en conclut que $f(X)$ est un ordonnancement au plus tôt.

□

Proposition 13. *Les ordonnancements au plus tôt sont des solutions dominantes pour les problèmes* Π_1 *et* Π_2.

Démonstration. Soit $Y = ((m_1, t_1), \ldots, (m_n, t_n)) \in \chi$ une solution qui n'est pas un ordonnancement au plus tôt. Il existe alors une solution

$$Y' = ((m_1, t_1), \ldots, (m_i, t'_i), \ldots, (m_n, t_n)) \in \chi$$

telle que $t_i = t'_i + K$, avec K un entier strictement positif, qui possède le même ordre d'exécution des tâches sur m_i et le même ordre d'exécution des tâches J_k telles que $\varphi_k = \varphi_i$. On note $f_1 : \chi \to \mathbb{R}$ la fonction objectif pour Π_1 à maximiser (nombre de plaquettes à produire dans un horizon fixé H) et $f_2 : \chi \to \mathbb{R}$ la

fonction objectif pour Π_2 à minimiser. Nous voulons montrer que $f_1(Y) < f_1(Y')$ et $f_2(Y') < f_2(Y)$.

La date de fin d'exécution de J_i dans Y est alors $C_i = t_i + \rho_{im_i}$ et dans Y' : $C'_i = t'_i + \rho_{im_i} < C_i$. Si $t_i \geq H + K$, alors on a $t'_i \geq H$ donc $f_1(Y) = f_1(Y')$, car le nombre de plaquettes produites ne change que pour J_i puisque rien n'est différent entre Y et Y' pour les autres tâches. Si $t_i \leq H - \rho_{im_i}$, alors on a $C'_i < C_i \leq H$ donc $f_1(Y) = f_1(Y')$. Si $H \geq t_i > H - \rho_{im_i}$, alors le nombre de plaquettes produites de J_i est de $\frac{H-t_i}{\rho_{im_i}}c_i$ dans Y et $\frac{H-t'_i}{\rho_{im_i}}c_i > \frac{H-t_i}{\rho_{im_i}}c_i$ dans Y'. D'où $f_1(Y) < f_1(Y')$. Enfin, si $H + K > t_i \geq H$, alors le nombre de plaquettes produites de J_i est de 0 dans Y et $\frac{H-t'_i}{\rho_{im_i}}c_i > 0$ (car $H - K \leq t'_i < H$) dans Y'. D'où $f_1(Y) < f_1(Y')$.

La date de fin d'exécution de J_i dans Y est alors $C_i = t_i + \rho_{im_i}$ et dans Y' : $C'_i = t'_i + \rho_{im_i} < C_i$. Ainsi, $w_iC_i > w_iC'_i$ d'où $f_2(Y') < f_2(Y)$. □

Théorème 6. Il existe dans $f(\mathcal{C})$ au moins une solution optimale pour Π_1 (resp. Π_2).

Démonstration. Soit $\chi_t \subset \chi$ l'ensemble des ordonnancements au plus tôt. On montre que $f(\mathcal{C}) = \chi_t$, ce qui avec la proposition 13 permet de conclure. L'inclusion $f(\mathcal{C}) \subset \chi_t$ se déduit de la proposition 12. Montrons que $\chi_t \subset f(\mathcal{C})$: Soit $Y = ((m_1, t_1), \ldots, (m_n, t_n)) \in \chi_t$ un ordonnancement au plus tôt. Soient $\sigma_1, \ldots, \sigma_n$ des indices dans $\{1, \ldots, n\}$ tels que : $t_{\sigma_1} \leq \ldots \leq t_{\sigma_n}$. Ainsi, l'individu $X = (\sigma_1, \ldots, \sigma_n, m_{\sigma_1}, \ldots, m_{\sigma_n}) \in \mathcal{C}$ vérifie $f(X) = Y$. En effet, dans l'algorithme 5, les tâches sont ordonnancées dans l'ordre donné par $\sigma_1, \ldots, \sigma_n$ et la date de début d'exécution se fait au plus tôt. Ainsi, dans $f(X)$, pour des tâches exécutées sur la même machine, l'ordre dans lequel elles sont exécutées est le même que dans Y sur chaque machine. Or, comme les dates de début d'exécution sont choisies au plus tôt, elles sont les mêmes. De plus, les tâches ayant la même ressource requise sont exécutées dans le même ordre, étant donné que leurs exécutions ne peuvent pas s'effectuer en parallèle. Leurs dates de début d'exécution sont donc les mêmes. Si t_i et t_j sont égaux, alors les tâches J_i et J_j ne sont pas exécutées sur la même machine et n'ont pas la même ressource auxiliaire requise, selon les contraintes. Par conséquent, la solution obtenue par l'algorithme sera la même, quel que soit l'ordre choisi pour i et j dans le codage. □

Ce résultat permet de justifier le codage utilisé, ainsi que l'algorithme permettant d'obtenir une solution du problème Π_1 (resp. Π_2). Dans ce qui suit, on

verra comment est construit l'algorithme mémétique, qui utilise ce codage afin de retourner une solution réalisable du problème Π_1 (resp. Π_2).

4.1.3 Schéma général

```
1  MÉMÉTIQUE(I)
2  Initialisation(P)
3  TriPopulationSelonFitness(P)
4  best ← P[1]      ▷ Meilleure solution trouvée
5  cbest ←Évaluation(I, best)
6  tant que Critère d'arrêt non atteint faire
7      pour i ← 1 à NombreDescendants faire
8          p1 ←Sélection(P)
9          p2 ←Sélection(P)
10         Desc[i] ←Croisement(p1, p2)
11         Desc[i] ←Recherche-Locale(I, Desc[i])
12     InsertionNouvelleGénération(P, Desc)
13     si P[1] meilleure que best alors
14         best ← P[1]      ▷ Actualisation de la meilleure solution
15         cbest ←Évaluation(I, best)
16 retourner DÉCODER(I, best)
```

Algorithme 6: Schéma général d'un algorithme mémétique

On présente ici les spécificités, détaillées dans les sections suivantes, de l'algorithme mémétique : initialisation et évaluation des individus, sélection, croisement, et recherche locale. L'algorithme 7 décrit les étapes de la méthode. Les arguments d'entrée sont : une instance I de Π_1 (resp. Π_2), la taille de la population N, le nombre d'itérations sans amélioration avant réinitialisation n_R, avant arrêt n_A, l'entier K pour l'étape de sélection, la fonction objectif ($f_{\text{obj}} = 1$ pour Π_1, $f_{\text{obj}} = 2$ pour Π_2), le taux t_D d'individus créés par croisement, le taux t_S d'individus retenus à l'itération suivante, le nombre ItR d'itérations de recherche locale, les valeurs positives p_1 et p_2 ($p_1 + p_2 \leq 100$), le réel $T > 0$, l'entier n_p et le réel $\alpha \in [0\ ,\ 1]$ utilisés

dans l'algorithme RECHERCHE-LOCALE, que l'on présentera dans la section 4.1.7.

```
1  POP-MÉMÉTIQUE(I, N, n_R, n_A, K, f_obj, t_D, t_S, ItR, p_1, p_2, T, n_p, α)
2  pour i ← 1 à N faire
3      P[i] ← INITIALISER(I)
4      P[i] ←RECHERCHE-LOCALE(I, P[i], ItR, f_obj, p_1, p_2, T, n_p, α)
5  nbItR ← 0, nbItA ← 0
6  RANGER-POPULATION(P, f_obj)
7  best ← P[1]      ▷ Meilleure solution trouvée
8  cbest ←ÉVALUATION(I, best, f_obj)
9  bestRein ← P[1], cbestRein ← cbest
10 tant que nbItA < n_A faire
11     pour i ← 1 à ⌊N × t_D⌋ faire
12         p_1 ←SÉLECTION(P, K)
13         p_2 ←SÉLECTION(P, K)
14         Desc[i] ←CROISEMENT(p_1, p_2)
15         Desc[i] ←RECHERCHE-LOCALE(I, Desc[i], ItR, f_obj, p_1, p_2, T, n_p, α)
16     RANGER-POPULATION(Desc, f_obj)      ▷ Tri avant insertion de la
           nouvelle génération
17     pour i ← 1 à ⌊⌊N × t_D⌋ × t_A⌋ faire
18         P[N − i + 1] ← Desc[i]
19     RANGER-POPULATION(P, f_obj)      ▷ Tri
20     evalact ←ÉVALUATION(I, P[1], f_obj)
21     si (f_obj = 1 et cbest < evalact) ou (f_obj = 2 et cbest > evalact) alors
22         best ← P[1]      ▷ Actualisation de la meilleure solution
23         cbest ← evalact , nbItA ← 0
24     sinon
25         nbItA ← nbItA + 1
26     si (f_obj = 1 et cbestRein < evalact) ou (f_obj = 2 et cbestRein > evalact)
       alors
27         bestRein ← P[1]      ▷ Actualisation de la meilleure solution
28         cbestRein ← evalact , nbItR ← 0
29     sinon
30         nbItR ← nbItR + 1
31     si nbItR = n_R alors
32         RÉINITIALISER(P)      ▷ Réinitialisation de la population
33         pour i ← 1 à N faire
34             P[i] ←RECHERCHE-LOCALE(I, P[i], ItR, f_obj, p_1, p_2, T, n_p, α)
35         nbItR ← 0
36         RANGER-POPULATION(P, f_obj)
37         bestRein ← P[1], cbestRein ←ÉVALUATION(I, bestRein, f_obj)
38 retourner DÉCODER(I, best)
```

Algorithme 7: Algorithme mémétique

Comme l'algorithme 7 l'indique, la taille de la population reste constante pendant la résolution. Une certaine proportion d'individus est sélectionnée (lignes 12 et 13) et créée par croisement (t_D) à la ligne 14, et dans la population, seuls les meilleurs individus (au sens d'une évaluation) seront retenus, en proportion t_S du nombre d'individus générés par croisement. Cette procédure de modification de la nouvelle population s'effectue aux lignes 17 et 18. Un compteur $nbItA$ est utilisé pour le nombre d'itérations sans améliorations par rapport à la meilleure solution trouvée depuis le début de l'algorithme ; il sert à définir le critère d'arrêt de l'algorithme. On l'actualise aux lignes 23 et 25. Lorsque $nbItA$ itérations sont exécutées sans améliorations, il s'arrête. Un autre compteur $nbItR$ (lignes 28 et 30) sert à la réinitialisation de la population lorsqu'il n'y a pas eu d'améliorations, et ce par rapport à la meilleure solution trouvée depuis la dernière réinitialisation. C'est pourquoi deux solutions sont conservées : (1) la meilleure solution trouvée et (2) la meilleure solution trouvée depuis la dernière réinitialisation.

4.1.4 Évaluation des individus

L'algorithme RANGER-POPULATION de la ligne 6 de l'algorithme 7 a pour simple effet de trier le tableau en entrée selon l'évaluation de la solution au sens de la fonction objectif désignée en entrée. Si $f_{\text{obj}} = 1$, les individus sont classés par ordre décroissant d'évaluation, sinon par ordre croissant, de sorte que la meilleure solution se trouve toujours à l'indice 1. Cette évaluation est présentée dans l'algorithme 8.

```
1  ÉVALUATION(I, (x_1, ..., x_n, y_1, ..., y_n), f_obj)
2  ((m_1, t_1), ..., (m_n, t_n)) ←DÉCODER(I, (x_1, ..., x_n, y_1, ..., y_n))
3  cout ← 0
4  si f_obj = 1 alors
5      pour i ← 1 à n faire
6          si t_i < H alors
7              cout ← cout + min(ρ_{im_i}, H − t_i) / ρ_{im_i} c_i
8  si f_obj = 2 alors
9      pour i ← 1 à n faire
10         cout ← cout + w_i(t_i + ρ_{im_i})
11 retourner cout
```

Algorithme 8: Évaluation d'un individu

L'évaluation est donc le coût de la solution correspondant à l'individu par l'application f. La section suivante explicite la procédure d'initialisation du codage de solutions dans l'algorithme.

4.1.5 Initialisation des individus

L'initialisation (tout comme la réinitialisation) se base sur un algorithme « randomisé », utilisant des tirages aléatoires pour générer un élément de l'ensemble $\mathcal{C}$. L'algorithme 9 permet de comprendre comment cette étape s'effectue. Dans les algorithmes de cette section, tout tirage aléatoire uniforme d'un nombre entier compris entre a et b est effectué par RANDOM(a, b).

```
1  INITIALISER(I)
2  pour i ← 1 à n faire
3      pris[i] ← faux        ▷ Indique si J_i est déjà placé
4  pour i ← 1 à n faire
5      k ←RANDOM(1, n − i + 1)        ▷ n − i + 1 est le nombre de tâches
                non encore placées
6      j ← 0
7      l ← 0
8      tant que j ≠ k faire
9          si pris[l + 1] = faux alors
10             j ← j + 1
11         l ← l + 1
12     x_i ← l
13     pris[l] ← vrai
14     k ←RANDOM(1, |M_l|)        ▷ Tirage aléatoire d'une machine
                qualifiée
15     j ← 0
16     l ← 0
17     tant que j ≠ k faire
18         si l + 1 ∈ M_{x_i} alors
19             j ← j + 1
20         l ← l + 1
21     y_i ← l
22 retourner (x_1, ..., x_n, y_1, ..., y_n)
```

Algorithme 9: Initialisation d'un individu

Remarque 6. La contrainte de qualification des machines (une tâche J_i ne peut être exécutée que sur une machine dont l'indice appartient à $\mathcal{M}_i$) doit être satisfaite à tout moment de l'exécution de l'algorithme, par tout individu, d'où cette vérification au moment de l'initialisation de la solution.

La réinitialisation, (algorithme RÉINITIALISER) s'effectue de la même façon. L'algorithme INITIALISER est utilisé N fois.

4.1.6 Sélection et croisement

Dans cette partie, les stratégies de sélection et de croisement sont présentées. Lorsqu'un individu est généré par sélection et croisement, une suite de modifications de cet individu est effectuée, afin d'en obtenir un autre avec une *bonne* évaluation.

Sélection L'algorithme 10 retourne un individu d'une population P donnée.

```
1 SÉLECTION(P, K)
2 r ← longueur[P] + 1
3 pour i ← 1 à K faire
4     c ←RANDOM(1, longueur[P])
5     si c < r alors
6         r ← c
7 retourner P[r]
```

Algorithme 10: Sélection d'un individu

L'individu retourné est le mieux évalué parmi K individus aléatoirement tirés (selon une loi uniforme).

Remarque 7. On suppose $K \geq 1$, d'où le fait que l'algorithme retourne forcément un individu de la population.

Dans l'algorithme 7, un couple d'individus est sélectionné (appel à l'algorithme SÉLECTION aux lignes 12 et 13) et un croisement s'opère ensuite sur eux (ligne 14) pour la génération d'un nouvel individu. Le croisement est détaillé par la suite.

Croisement L'algorithme 11 décrit la manière dont se fait le croisement entre deux individus $u = (x_1, \ldots, x_n, y_1, \ldots, y_n)$ et $u' = (x'_1, \ldots, x'_n, y'_1, \ldots, y'_n)$.

```
1  CROISEMENT((x_1, ..., x_n, y_1, ..., y_n), (x'_1, ..., x'_n, y'_1, ..., y'_n))
2  pour i ← 1 à n faire
3      pris[i] ← faux
4      v[i] ←RANDOM(0, 1)      ▷ Construction du vecteur qui dessine
                                 le motif du croisement
5  k_1 ← 1
6  pour i ← 1 à n faire
7      si g[i] = 1 alors
8          x''_{k_1} ← x_i      ▷ Les composantes à 1 indiquent quelles
                                 parties de u on retient
9          y''_{k_1} ← y_i
10         k_1 ← k_1 + 1
11         pris[x_i] ← vrai     ▷ Pour ne pas mettre deux fois la même
                                 tâche dans le résultat
12 pour i ← 1 à n faire
13     si pris[x'_i] = faux alors
14         x''_{k_1} ← x'_i     ▷ Les composantes restantes à prendre dans u'
15         y''_{k_1} ← y'_i
16         k_1 ← k_1 + 1
17 retourner (x''_1, ..., x''_n, y''_1, ..., y''_n)
```

Algorithme 11: Croisement de deux individus

Le principe de l'algorithme est de tirer aléatoirement un vecteur

$$v = (v_1, \ldots, v_n) \in \{0, 1\}^n$$

qui déterminera les n_1 indices qui formeront (dans le même ordre que celui de u) les n_1 premières valeurs dans l'individu retourné. Les autres valeurs seront dans le même ordre que dans u' aux $n - n_1$ derniers indices. Dans l'individu $u'' = (x''_1, \ldots, x''_n, y''_1, \ldots, y''_n)$ retourné, pour tout $i \in \{1, \ldots, n_1\}$, si $x''_i = x_k$, alors $y''_i = y_k$, et pour tout $i \in \{n_1 + 1, \ldots, n\}$, si $x''_i = x'_k$, alors $y''_i = y'_k$.

Exemple 1. *Si l'on a* $n = 5$, $v = \{1, 0, 1, 0, 1\}$ *et les individus suivants à croiser :*

$$u = (x_1, \ldots, x_5, y_1, \ldots, y_5) = (\mathbf{4}, 3, \mathbf{2}, 5, \mathbf{1}, \mathbf{1}, 1, \mathbf{2}, 1, \mathbf{2}) \tag{4.1}$$

$$et\ u' = (x'_1, \ldots, x'_5, y'_1, \ldots, y'_5) = (5, 4, 1, 3, 2, 1, 2, 2, 1, 2), \tag{4.2}$$

alors $u'' = (4, 2, 1, 5, 3, 1, 2, 2, 1, 1)$.

Les indices de tâches 4, 2 *et* 1 *sont ceux qui correspondent dans* u *aux composantes de* v *égales à* 1. *Ainsi, ici* $n_1 = 3$, *on retrouve ces valeurs aux* 3 *premières composantes de* u'' *et les indices des machines pour ces tâches restent les mêmes dans* u''. *Les indices restants sont* 5 *et* 3 *qui seront dans* u'' *dans le même ordre que celui dans lequel ils apparaissent dans* u'. *D'où le vecteur obtenu.*

4.1.7 Recherche locale et voisinages

On termine la description de la méthode de résolution par l'explication de la stratégie de recherche locale qui a été retenue. L'algorithme 12 montre comment un individu généré (que ce soit par croisement ou au moment de l'initialisation, mais aussi à la réinitialisation de la population) est *muté* par un procédé qui vise à améliorer la qualité de la solution correspondante.

On considère dans cet algorithme que RAND() est un réel compris entre 0 et 1,

tiré aléatoirement selon une loi uniforme.

```
1  RECHERCHE-LOCALE(I, u, ItR, f_obj, p_1, p_2, T, n_p, α)
2  c ←ÉVALUATION(I, u, f_obj)
3  best ← u        ▷ Pour conserver la meilleure solution obtenue
4  cbest ← c
5  pour i ← 1 à ItR faire
6      p ←RANDOM(1, 100)      ▷ Choix de l'opérateur de voisinage
                par tirage aléatoire
7      si p ≤ p_1 alors
8          u' ←VOISINAGE-V1(I, u)
9      sinon si p ≤ p_1 + p_2 alors
10         u' ←VOISINAGE-V2(I, u)
11     sinon
12         u' ←VOISINAGE-V3(I, u)
13     c' ←ÉVALUATION(I, u', f_obj)
14     si (f_obj = 1 et c < c') ou (f_obj = 2 et c > c') alors
15         c ← c'
16         u ← u'
17     sinon
18         Δ ← |c − c'|
19         p ←RAND()
20         si p < exp^(−Δ/T) alors
21             c ← c'
22             u ← u'
23     si i ≡ 0 mod n_p alors
24         T ← αT      ▷ Diminution de T toutes les n_p itérations
25     si (f_obj = 1 et cbest < c) ou (f_obj = 2 et cbest > c) alors
26         cbest ← c      ▷ Pour conserver la meilleure solution
                    trouvée
27         best ← u
28 retourner best
```

Algorithme 12: Recherche locale pour améliorer la qualité d'une solution

Pendant ItR itérations, un individu $u \in \mathcal{C}$ est modifié, au moyen de trois voisinages V_1, V_2 et V_3 (respectivement utilisés dans les algorithmes VOISINAGE-V1, VOISINAGE-V2 et VOISINAGE-V3). À chaque itération, l'un de ces trois voisinages est tiré aléatoirement : le voisinage V_1 avec la probabilité $\frac{p_1}{100}$, le voisinage V_2 avec la probabilité $\frac{p_2}{100}$ et le voisinage V_3 avec la probabilité $1 - \frac{p_1+p_2}{100}$. L'individu $u' \in \mathcal{C}$ obtenu est conservé s'il a une meilleure évaluation (ligne 14). S'il a une moins bonne

évaluation, il sera conservé avec une probabilité de $\exp^{-\frac{\Delta}{T}}$, où T est un paramètre de l'algorithme et Δ est la valeur absolue de la différence entre les évaluations de u et u'. Au fil des itérations de l'algorithme, la valeur de T est diminuée (toutes les n_p itérations, par multiplication par α, $0 \leq \alpha \leq 1$), de sorte que la probabilité de conserver une solution u' de moins bonne évaluation diminue. L'individu retourné est celui, parmi les individus calculés, qui aura la meilleure évaluation.

Décrivons maintenant les trois opérateurs de voisinage V_1, V_2 et V_3 retenus.

$$V_1 : \mathcal{C} \times \{1, \ldots, n\} \times \{1, \ldots, m\} \to \mathcal{C}$$
$$((x_1, \ldots, x_i, \ldots, x_n, y_1, \ldots, y_i, \ldots, y_n), i, j) \mapsto (x_1, \ldots, x_i, \ldots, x_n, y_1, \ldots, j, \ldots, y_n).$$

Le voisinage $V_1(u, i, j)$ correspond au remplacement d'une composante (parmi les n dernières) par une valeur $y \in \{1, \ldots, m\}$. C'est le voisinage de changement de machine affectée.

Exemple 2. *Pour $n = 4$, $m = 3$, on a*

$$V_1((1, \mathbf{4}, 3, 2, 1, \mathbf{1}, 1, 1), 2, 3) = (1, \mathbf{4}, 3, 2, 1, \mathbf{3}, 1, 1).$$

$$V_2 : \mathcal{C} \times \{1, \ldots, n\} \times \{1, \ldots, n+1\} \to \mathcal{C}$$
$$((x_1, \ldots, x_i, \ldots, x_j, \ldots, x_n, y_1, \ldots, y_i, \ldots, y_j, \ldots, y_n), i, j)$$
$$\mapsto$$
$$(x_1, \ldots, x_{i-1}, x_{i+1} \ldots, x_{j-1}, x_i, x_j, \ldots, x_n, y_1, \ldots, y_{i-1}, y_{i+1}, \ldots, y_{j-1}, y_i, y_j, \ldots, y_n).$$

Le voisinage $V_2(u, i, j)$ correspond donc au déplacement de la i-ème (resp. la $n+i$-ème) composante de u à l'indice correspondant à la composante précédant la j-ème (resp. la $n+j$-ème) dans u. Lorsque $j = 1$, le nouvel indice de la composante déplacée est 1 (resp. $n+1$). Lorsque $j = n+1$, son nouvel indice est n (resp. $2n$). C'est le voisinage de déplacement d'une tâche.

Exemple 3. *Pour $n = 4$, $m = 3$, on a*

$$V_2((1, \mathbf{4}, 3, 2, 1, \mathbf{2}, 1, 1), 2, 5) = (1, 3, 2, \mathbf{4}, 1, 1, 1, \mathbf{2}),$$

ou encore

$$V_2((1, \mathbf{4}, 3, 2, 1, \mathbf{2}, 1, 1), 2, 4) = (1, 3, \mathbf{4}, 2, 1, 1, \mathbf{2}, 1)$$

et

$$V_2((1,\mathbf{4},3,2,1,\mathbf{2},1,1),2,1) = (\mathbf{4},1,3,2,\mathbf{2},1,1,1).$$

$$\begin{gathered} V_3 : \mathcal{C} \times \{1,\dots,n\} \times \{1,\dots,n\} \to \mathcal{C} \\ ((x_1,\dots,x_i,\dots,x_j,\dots,x_n,y_1,\dots,y_i,\dots,y_j,\dots,y_n),i,j) \\ \mapsto \\ (x_1,\dots,x_j\dots,x_i\dots,x_n,y_1,\dots,y_j,\dots,y_i,\dots,y_n). \end{gathered}$$

Le voisinage $V_3(u,i,j)$ correspond donc à l'échange de deux couples de composantes (la i-ème et la j-ème, la $n+i$-ème et la $n+j$-ème). C'est le voisinage d'échange de tâches.

Exemple 4. *Toujours pour $n = 4$, $m = 3$, on a*

$$V_3((1,\mathbf{4},3,\mathbf{2},1,\mathbf{2},1,\mathbf{1}),2,4) = (1,\mathbf{2},3,\mathbf{4},1,\mathbf{1},1,\mathbf{2}),$$

En s'appuyant sur ces définitions, on conçoit les algorithmes 13, 14 et 15.

```
1  VOISINAGE-V1(I, (x_1, ..., x_n, y_1, ..., y_n))
2  i ←RANDOM(1, n)        ▷ Indice sur lequel la modification
                            s'applique
3  j ←RANDOM(1, |M_{x_i}|)    ▷ indice de la machine : elle doit être
                                qualifiée pour x_i
4  y ← 0
5  l ← 0
6  tant que y < j faire
7      si l + 1 ∈ M_{x_i} alors
8          y ← y + 1
9      l ← l + 1
10 retourner V_1((x_1, ..., x_n, y_1, ..., y_n), i, y)
```

Algorithme 13: Voisinage V_1 de changement de machine affectée

Remarque 8. On veille, dans l'algorithme VOISINAGE-V1 (algorithme 13), à ce que l'individu retourné appartienne bien à l'ensemble $\mathcal{C}$ par un tirage aléatoire du nouvel indice de machine dans l'ensemble des indices des machines qualifiées pour la tâche correspondante.

```
1 VOISINAGE-V2(I, u)
2 i ←RANDOM(1, n)        ▷ Indice de la composante à déplacer
3 j ←RANDOM(1, n + 1)
4 retourner V_2(u, i, j)
```

Algorithme 14: Voisinage V_2 de déplacement d'une tâche

```
1 VOISINAGE-V3(I, u)
2 i ←RANDOM(1, n)        ▷ Indices des composantes à échanger
3 j ←RANDOM(1, n)
4 retourner V_3(u, i, j)
```

Algorithme 15: Voisinage V_3 d'échange de tâches

Définition 5. Soit $u = (x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$. Deux composantes x_i et x_j de u sont dites *liées par rapport à* u si et seulement si l'une des deux conditions suivantes est vérifiée :

1. Les tâches J_{x_i} et J_{x_j} ont la même ressource auxiliaire requise : $\varphi_{x_i} = \varphi_{x_j}$.
2. Les tâches J_{x_i} et J_{x_j} sont exécutées sur la même machine dans la solution $f(u) : y_i = y_j$.

Cette définition s'applique à un ensemble de composantes $\{x_{i_1}, \ldots, x_{i_s}\}$ qui sont liées entre elles si et seulement si l'une des conditions suivantes est vérifiée :

1. Les tâches $J_{x_{i_1}}, \ldots, J_{x_{i_s}}$ ont la même ressource auxiliaire requise : $\varphi_{x_{i_1}} = \ldots = \varphi_{x_{i_s}}$.
2. Les tâches $J_{x_{i_1}}, \ldots, J_{x_{i_s}}$ sont exécutées sur la même machine dans la solution $f(u) : y_{i_1} = \ldots = y_{i_s}$.

Proposition 14. *On a, pour tout* $u \in \mathcal{C}$ *et pour tout couple* $(i, j) \in \{1, \ldots, n\}^2$, $i \leq j$,

$$V_3(u, i, j) = V_2(V_2(u, i, j), j, i).$$

Démonstration. La composante b (resp. a) d'indice j (resp. i) dans u sera d'indice j (resp. $j-1$) dans $u' = V_2(u, i, j)$. Comme $i \leq j$ dans $V_2(u', j, i)$, la composante a aura l'indice j et la composante b aura l'indice i. Les autres composantes conservent le même indice que dans u (celles d'indice inférieur à i ou supérieur à j ne changent jamais, celles d'indice entre i et j voient leur indice diminuer d'une unité dans u' et augmenter d'une unité dans $V_2(u', j, i)$). □

Ce résultat permet de voir que le voisinage V_3 peut s'écrire sous la forme d'une composition de deux applications du voisinage V_2.

4.1.8 Propriétés de l'algorithme mémétique

Dans ce qui suit, on apporte une justification des voisinages utilisés dans la méthode de résolution. En particulier, on montre qu'ils permettent d'atteindre une solution optimale.

Proposition 15. *Pour tout couple $(u_1, u_2) \in \mathcal{C}^2$ d'individus, il existe des entiers $i_1, \ldots, i_s$, $l_1, \ldots, l_s$, $j_1, \ldots, j_r$, $k_1, \ldots, k_r$ tels que*

$$u_2 = V_2(\ldots, (V_2(V_1(\ldots (V_1(u_2, j_1, k_1), \ldots), j_r, k_r), i_1, l_1), \ldots), i_s, l_s).$$

Autrement dit, il existe une suite d'opérations de voisinages permettant d'obtenir u_2 à partir de u_1.

Démonstration. On rappelle d'abord un résultat de la théorie des groupes.

Soit n un entier naturel et E l'ensemble $\{1, \ldots, n\}$. Le groupe symétrique de E est l'ensemble S_n des permutations de E, c'est-à-dire les bijections de E dans lui-même. Il est muni de la loi de composition d'applications $\circ$. Une *transposition*, ou un 2-cycle, est une permutation qui échange deux éléments et laisse tous les autres inchangés. L'ensemble des transpositions appartient bien à S_n et toute permutation de S_n peut s'obtenir par la composition d'un nombre fini de transpositions. L'algorithme de construction de ces compositions se décrit ainsi :

Soit σ une permutation quelconque de S_n. Si c'est la permutation identité, l'algorithme s'arrête puisqu'on obtient le résultat voulu. Sinon, on note k le premier point non fixe de σ. On compose alors à σ la transposition t_1 qui échange k et $\sigma(k)$. Ainsi, dans $t_1 \circ \sigma$, tous les points de 1 à k sont des points fixes. On poursuit de même avec les points de $k+1$ à n. On obtient de cette manière une composition d'un nombre fini de transpositions qui, composée à σ, est égale à l'application identité. Il suffit alors d'utiliser l'égalité, vraie pour toutes bijections α,β, $(\alpha \circ \beta)^{-1} = \beta^{-1} \circ \alpha^{-1}$.

Soient $u_1 = (x_1, \ldots, x_n, y_1, \ldots, y_n)$ et $u_2 = (x'_1, \ldots, x'_n, y'_1, \ldots, y'_n)$ deux éléments de $\mathcal{C}$. Par définition de $\mathcal{C}$, on a $\{x_1, \ldots, x_n\} = \{x'_1, \ldots, x'_n\} = \{1, \ldots, n\}$. Pour tout $i \in \{1, \ldots, n\}$, notons par σ_i l'indice tel que $x'_{\sigma_i} = x_i$.

Ainsi, on considère l'élément

$$u_3 = V_1(\ldots (V_1(u_1, 1, y'_{\sigma_1}), \ldots), n, y'_{\sigma_n}) = (x_1, \ldots, x_n, y'_{\sigma_1}, \ldots, y'_{\sigma_n}).$$

De plus, on sait par le résultat de théorie des groupes rappelé au début de la démonstration, qu'il existe des indices $i_1, \ldots, i_s$, $j_1, \ldots, j_s$ tels que

$$u_2 = (x'_1, \ldots, x'_n, y'_1, \ldots, y'_n) = V_3(\ldots(V_3(u_3, i_1, j_1), \ldots), i_s, j_s).$$

Ceci implique, d'après la proposition 14, qu'il existe des indices $i'_1, \ldots, i'_{2s}$, $j'_1, \ldots, j'_{2s}$ tels que :

$$u_2 = (x'_1, \ldots, x'_n, y'_1, \ldots, y'_n) = V_2(\ldots(V_2(u_3, i'_1, j'_1), \ldots), i'_{2s}, j'_{2s}),$$

ce qui achève la démonstration.

□

Ce résultat est intéressant au sens où il permet de fournir une preuve au théorème suivant.

Théorème 7. Il existe une solution optimale dans χ susceptible d'être retournée par l'algorithme POP-Mémétique, pour le problème Π_1 (resp. Π_2).

Démonstration. Ce théorème se déduit directement de trois assertions :

1. Le théorème 6 qui affirme qu'il existe $y \in \mathcal{C}$ tel que $f(y)$ est optimale pour Π_1 (resp. Π_2).
2. L'algorithme POP-Mémétique (par la procédure Recherche-Locale) est conçu de manière à garantir une probabilité non-nulle de passer d'un individu à un autre par les opérateurs de voisinage.
3. La proposition 15 qui montre que tout élément $u \in \mathcal{C}$ peut être atteint par la méthode de recherche locale.

□

On vient ainsi de démontrer la *connexité* des voisinages de la méthode. Ainsi, il est possible, par ce qui précède, d'aboutir à n'importe quelle solution dominante de χ (l'ensemble des ordonnancements au plus tôt).

On remarque que l'application f (qui, on le rappelle, associe à tout individu une solution dans χ, qui est un ordonnancement au plus tôt) n'est pas injective. Autrement dit, plusieurs individus peuvent avoir la même image par f.

Définition 6. Soit $u \in \mathcal{C}$. On appelle *voisin significatif* tout $u' \in \mathcal{C}$ tel que $u' = V_i(u, a, b)$ $(i \in \{1, 2, 3\})$ et $f(u) \neq f(u')$.

On peut utiliser la propriété suivante dans l'algorithme mémétique.

Proposition 16. *Soient* $u = (x_1, \ldots, x_n, y_1, \ldots, y_n) \in \mathcal{C}$ *et* $i \in \{1, \ldots, n\}$*, et soient* i_1 *et* i_2 *deux éléments de* $\{1, \ldots, n\}$ *tels que* $i_1 \leq i \leq i_2$ *et pour tout* $k \in \{i_1, \ldots, i_2\} \setminus \{i\}$*,* $y_k \neq y_i$ *et* $\varphi_{x_k} \neq \varphi_{x_i}$*.*

Alors on a, pour tout $k \in \{i_1, \ldots, i_2\}$*,*

$$f(V_2(u, i, k)) = f(u).$$

Démonstration. Par définition de i_1 et i_2, et par l'algorithme 5, l'ordre des indices liés entre eux par rapport à u est le même dans u et $V_2(u, i, k)$.

Les dates de début d'exécution sont calculées au plus tôt dans l'algorithme 5, de sorte à respecter l'ordre d'exécution des tâches sur une même machine ainsi que l'ordre d'exécution des tâches nécessitant la même ressource auxiliaire, et les contraintes (setup, transfert de ressource, non-simultanéité d'exécution de tâches nécessitant la même ressource, non-simultanéité d'exécution de tâches sur une même machine).

Ainsi, l'algorithme DÉCODER retourne les mêmes solutions :

$$f(V_2(u, i, k)) = f(u).$$

□

En effet, la ligne 3 de l'algorithme 14 peut être modifiée de sorte à ajouter cette vérification dans le choix de l'indice vers lequel déplacer une tâche. On peut ainsi calculer, pour $u \in \mathcal{C}$ et $i \in \{1, \ldots, n\}$ donnés, les indices i_1 et i_2 tels que $i_1 \leq i \leq i_2$, pour tout $k \in \{i_1, \ldots, i_2\} \setminus \{i\}$, $y_k \neq y_i$ et $\varphi_{x_k} \neq \varphi_{x_i}$ et, si $i_1 > 1$, $y_{i_1-1} = y_i$ ou $\varphi_{x_{i_1-1}} = \varphi_{x_i}$ et si $i_2 < n$, $y_{i_2+1} = y_i$ ou $\varphi_{x_{i_2+1}} = \varphi_{x_i}$.

Ainsi la recherche de l'indice vers lequel déplacer une tâche se fait par tirage aléatoire d'un indice dans $\{1, \ldots, i_1 - 1\} \cup \{i_2 + 1, \ldots, n\}$.

4.1.9 Résultats numériques

Des expérimentations ont été menées pour analyser la performance de l'algorithme mémétique, sur un processeur Intel Core i7-2640M CPU, 2.80GHz, sur la base d'une implémentation en langage C++. Ces essais ont été réalisés sur des instances générées aléatoirement. Ces instances sont décrites selon un format $n - m - \ell - H$, où n est le nombre de tâches, m le nombre de machines, ℓ le nombre de ressources auxiliaires et H l'horizon de temps pour la prise en compte

des plaquettes. Dix instances de chaque type sont générées comme suit (tous les tirages aléatoires suivent une loi uniforme).

- Les durées d'exécution sont générées aléatoirement dans $\{1, \ldots, 10\}$.
- Les temps de setup sont générés aléatoirement dans $\{1, \ldots, 5\}$.
- Pour chaque tâche, son nombre de machines qualifiées est choisi aléatoirement (entre 1 et m) et l'ensemble des machines qualifiées est tiré aléatoirement.
- Pour les ℓ premières tâches, la ressource auxiliaire requise de la i-ème tâche est la i-ème ressource. Pour les autres tâches, la ressource requise est tirée aléatoirement.
- Le nombre de plaquettes de chaque tâche est de 25 avec une probabilité de 90%, et un nombre aléatoirement fixé entre 1 et 25 avec une probabilité de 10%.
- Les priorités des tâches sont fixées aléatoirement dans $\{1, \ldots, 10\}$.
- La position initiale de chaque ressource auxiliaire (masque) est aléatoirement fixée dans $\{0, \ldots, m\}$.

La taille de la population et la façon de paramétrer la fréquence des voisinages de recherche locale ont un impact sur la qualité des solutions obtenues. Quelques expérimentations ont été menées dans ce sens. Les autres paramètres de l'algorithme mémétique sont fixés à des valeurs standard, reconnues comme généralement efficaces dans la littérature sur les algorithmes génétiques.

- $t_D = 50\%$: les individus fils sont générés en quantité égale à la moitié de la population ;
- $t_S = 80\%$: parmi les fils restants, les meilleurs 80% sont conservés. En effet, on ne veut pas trop bouleverser la population ;
- $K = \frac{N}{4}$: le nombre d'individus parmi lesquels on sélectionne en vue du croisement. Cette valeur, si elle est trop élevée, peut empêcher certains individus intéressants d'être sélectionnés ;
- $\alpha = 0.9$, $T = 100$, $ItR = 100$, $n_p = 5$: paramètres de la procédure de recuit simulé ;
- $n_A = \min(\max(500, 3n), 800)$: nombre d'itérations sans améliorations avant arrêt de l'algorithme ; doit être proportionnel à n mais les bornes 500 et 800 sont là pour garantir une étendue correcte ;
- $n_R = \frac{n_A}{5}$: nombre d'itérations sans amélioration avant réinitialisation de la population ; doît être inférieur à n_A pour garantir certaines réinitialisations.

Les résultats présentés aux tables 4.1 et 4.2 montrent que, pour Π_1 et Π_2, les meilleures solutions sont obtenues pour $N = 100$ dans la plupart des cas. En effet, avec 30 individus, la diversité de la population est limitée. Il est préférable d'utiliser au moins deux fois plus d'individus. Mais la durée d'exécution de l'algorithme augmente avec la taille de la population, ce qui empêche d'augmenter ce paramètre

	N		
$(n-m-\ell-H)$	30	60	100
$10-2-3-25$	**206** ($5s$)	**206** $10s$	**206** ($16s$)
$20-2-3-50$	439 ($14s$)	441 ($26s$)	**442** ($36s$)
$30-3-4-50$	643 ($30s$)	647 ($65s$)	**654** ($102s$)
$40-3-4-60$	791 ($64s$)	789 ($94s$)	**793** ($188s$)
$50-3-6-80$	1052 ($53s$)	1071 ($190s$)	**1078** ($450s$)
$70-4-10-85$	1551 ($90s$)	1572 ($236s$)	**1586** ($346s$)
$90-4-10-110$	1967 ($209s$)	1982 ($319s$)	**2037** ($437s$)
$100-5-12-100$	2166 ($301s$)	2218 ($397s$)	**2257** ($522s$)
$120-5-12-120$	2512 ($340s$)	2650 ($415s$)	**2707** ($663s$)
$150-6-14-125$	3215 ($244s$)	**3406** ($521s$)	3397 ($943s$)
$200-8-20-100$	3833 ($323s$)	3962 ($674s$)	**4000** ($783s$)

TABLE 4.1 – Qualité des solutions pour Π_1 ($\max \sum_{i=1}^{n} c_i\theta_i$) selon la taille de la population($p_1 = 45\%$, $p_2 = 45\%$).

à volonté. Par exemple, il n'est pas nécessaire d'excéder 60 pour Π_1 puisque, pour certaines instances, les solutions sont même meilleures qu'avec $N = 100$, et pour les autres instances, l'écart n'est pas très important.

Les tables 4.3 et 4.4 montrent l'impact de la fréquence d'utilisation des différents voisinages. Pour Π_1, les meilleures solutions pour plus de 90% des instances sont obtenues avec $p_1 = 80\%$ et $p_2 = 10\%$. Quand le nombre de plaquettes traitées avant un horizon de temps H est maximisé, il est très important de privilégier la réaffectation d'une tâche sur une autre machine qualifiée. La probabilité du voisinage correspondant doit donc être élevée. La tendance est toute autre concernant Π^2 : $(10\%, 80\%)$ et $(45\%, 45\%)$ sont plus efficaces, puisque la plupart des instances sont résolues à la meilleure valeur connue quand $p_2 \geq 45\%$. Le voisinage de reséquencement est alors plus efficace pour Π_2, i.e. quand la somme pondérée des dates de fin d'exécution est minimisée. Ces résultats illustrent le fait que le meilleur paramétrage diffère selon la fonction objectif.

Pour analyser l'impact du test supplémentaire basé sur la proposition 16, la table 4.5 montre le pourcentage moyen d'indices interdits pour une tâche donnée quand un voisinage de reséquencement s'opère. Cette valeur est la moyenne de tous les ratios $\frac{i_2-i_1+1}{n}$ calculés en un appel de l'algorithme mémétique. Ce pourcentage décroît avec la taille de l'instance, ce qui est cohérent puisqu'avec plus de machines

	N		
$(n-m-\ell-H)$	30	60	100
$10-2-3-25$	**156** ($5s$)	**156** ($10s$)	**156** ($16s$)
$20-2-3-50$	529 ($11s$)	529 ($21s$)	**526** ($29s$)
$30-3-4-50$	747 ($36s$)	**746** ($60s$)	750 ($101s$)
$40-3-4-60$	1391 ($47s$)	1386 ($94s$)	**1385** ($130s$)
$50-3-6-80$	1926 ($59s$)	1841 ($148s$)	**1840** ($389s$)
$70-4-10-85$	2488 ($105s$)	2523 ($212s$)	**2462** ($490s$)
$90-4-10-110$	3953 ($187s$)	3817 ($293s$)	**3743** ($532s$)
$100-5-12-100$	4276 ($256s$)	4172 ($513s$)	**4134** ($845s$)
$120-5-12-120$	6196 ($220s$)	5922 ($418s$)	**5861** ($417s$)
$150-6-14-125$	7266 ($360s$)	7370 ($566s$)	**7125** ($850s$)
$200-8-20-100$	8980 ($402s$)	8960 ($525s$)	**8755** ($855s$)

TABLE 4.2 – Qualité des solutions pour Π_2 ($\min \sum_{i=1}^{n} w_i C_i$) selon la taille de la population ($p_1 = 45\%$, $p_2 = 45\%$).

	(p_1, p_2)		
$(n-m-\ell-H)$	$(10\%, 80\%)$	$(45\%, 45\%)$	$(80\%, 10\%)$
$10-2-3-25$	**206**	**206**	**206**
$20-2-3-50$	425	441	**453**
$30-3-4-50$	636	654	**659**
$40-3-4-60$	795	793	**809**
$50-3-6-80$	1080	1078	**1085**
$70-4-10-85$	1581	**1586**	1581
$90-4-10-110$	2031	2037	**2095**
$100-5-12-100$	2256	2257	**2284**
$120-5-12-120$	2701	2707	**2711**
$150-6-14-125$	3337	3397	**3412**
$200-8-20-100$	3989	4000	**4019**

TABLE 4.3 – Qualité des solutions pour Π_1 ($\max \sum_{i=1}^{n} c_i \theta_i$) selon le voisinage ($N = 100$).

	(p_1, p_2)		
$(n-m-\ell-H)$	$(10\%, 80\%)$	$(45\%, 45\%)$	$(80\%, 10\%)$
$10-2-3-25$	**156**	**156**	**156**
$20-2-3-50$	541	**526**	563
$30-3-4-50$	778	**750**	788
$40-3-4-60$	**1377**	1385	1395
$50-3-6-80$	1853	**1840**	1851
$70-4-10-85$	**2445**	2462	2462
$90-4-10-110$	**3754**	3770	3793
$100-5-12-100$	**4128**	4134	4178
$120-5-12-120$	5865	**5861**	5869
$150-6-14-125$	7220	**7125**	7182
$200-8-20-100$	8811	**8755**	8816

TABLE 4.4 – Qualité des solutions pour Π_2 ($\min \sum_{i=1}^{n} w_i C_i$) selon le voisinage ($N = 100$).

et plus de ressources auxiliaires, il y a moins de chance de voir beaucoup de tâches partager la même machine ou la même ressource auxiliaire. Pour 200 tâches, la moyenne atteint 7%, ce qui correspond à environ 14 indices interdits en moyenne à chaque fois que la règle est appliquée. Cela est intéressant car des voisins inutiles sont évités.

4.2 Améliorations de l'algorithme mémétique

Plusieurs tests menés sur des instances de grande taille ont révélé que l'importance de la méthode de recherche locale (algorithme de recuit simulé présenté à la section 4.1.7) devait augmenter avec la taille de l'instance. Lorsqu'il est bien paramétré et qu'il prend le pas sur la gestion de la population, cet algorithme permet d'obtenir des solutions bien meilleures. En effet, sur les instances à 200 tâches, on observe qu'avec un seul individu dans la population et un nombre d'itérations proche de 3.10^7, on obtient une amélioration de la qualité de la solution de 20% pour un temps de résolution 4 fois inférieur.

Avant de présenter ces résultats de façon plus détaillée, on décrit la modification apportée à l'algorithme de recherche locale, en lui ajoutant trois paramètres :

$(n-m-\ell)$	% moyen des indices interdits
$25-2-3$	18%
$50-3-6$	12%
$90-4-10$	9%
$120-5-12$	8%
$200-8-20$	7%

TABLE 4.5 – Pourcentage (moyenne sur 10 instances) des indices interdits quand la règle de la proposition 16 est appliquée.

1. Paramètre ζ_1 : nombre d'itérations consécutives sans amélioration de la solution courante, avant de *redémarrer* la température. L'idée est de redonner la possibilité de dégrader la solution lorsqu'un optimum local empêche de poursuivre la descente.
2. Paramètre ζ_2 : nombre d'itérations consécutives sans amélioration de la meilleure solution, avant de diminuer la température de redémarrage. En effet, ce paramètre est essentiel car on observe qu'en conservant la même valeur à chaque redémarrage de température, on dégrade trop la solution, ce qui augmente le temps de convergence vers une meilleure solution. Ainsi, plus on a de difficultés à dépasser la meilleure solution connue, moins on s'autorise à dégrader la solution. D'où la définition d'un palier de convergence pour réduire la valeur de la température initiale au fil des itérations.
3. Paramètre β : coefficient (réel compris entre 0 et 1) de diminution de la température de redémarrage.

L'algorithme 16 est l'algorithme de recherche locale modifié de sorte à prendre en compte les trois paramètres ζ_1, ζ_2 et β. Il dérive directement de l'algorithme 12. Les modifications portent sur la mise à jour du paramètre de température T à l'aide des trois paramètres supplémentaires ζ_1, ζ_2 et β. La variable n_1 est un compteur incrémenté (ligne 17) lorsqu'il n'y a pas amélioration de la solution courante et réinitialisée (ligne 15) quand la solution courante est améliorée. Lorsqu'elle atteint ζ_1, on redémarre la température (ligne 28). La variable n_2 est un compteur incrémenté (ligne 26) lorsqu'il n'y a pas amélioration de la meilleure solution connue et réinitialisée (ligne 24) quand la meilleure solution connue est actualisée. Lorsqu'elle atteint ζ_2, on diminue la température de redémarrage (ligne 30).

Pour chacun des 11 ensembles de 10 instances résolues dans les tests précédents, une exécution limitée à 120 secondes de résolution a été lancée. Les résultats sont consignés dans la table 4.6. On observe que les résultats obtenus sont de loin les meilleurs.

```
1  RECUIT-SIMULÉ(I, u, ItR, f_obj, p_1, p_2, T, n_p, α, ζ_1, ζ_2, β)
2  c ←ÉVALUATION(I, u, f_obj)
3  best ← u, cbest ← c
4  n_1 ← 0, n_2 ← 0, T_0 ← T
5  pour i ← 1 à ItR faire
6      p ←RANDOM(1, 100)
7      si p ≤ p_1 alors
8          u' ←VOISINAGE-V1(I, u)
9      sinon si p ≤ p_1 + p_2 alors
10         u' ←VOISINAGE-V2(I, u)
11     sinon
12         u' ←VOISINAGE-V3(I, u)
13     c' ←ÉVALUATION(I, u', f_obj)
14     si (f_obj = 1 et c < c') ou (f_obj = 2 et c > c') alors
15         c ← c', u ← u', n_1 ← 0
16     sinon
17         n_1 ← n_1 + 1
18         Δ ← |c − c'|, p ←RAND()
19         si p < exp^{−Δ/T} alors
20             c ← c', u ← u'
21     si i ≡ 0 mod n_p alors
22         T ← αT        ▷ Diminution de T toutes les n_p itérations
23     si (f_obj = 1 et cbest < c) ou (f_obj = 2 et cbest > c) alors
24         cbest ← c, best ← u, n_2 ← 0
25     sinon
26         n_2 ← n_2 + 1
27     si n_1 = ζ_1 alors
28         n_1 ← 0, T ← T_0
29     si n_2 = ζ_2 alors
30         n_2 ← 0, T_0 ← βT_0
31 retourner best
```

Algorithme 16: Recherche locale, recuit simulé avec température adaptative

< 120s	Moyenne	Instances									
		1	2	3	4	5	6	7	8	9	10
$10-2-3$	155,5	150	150	142	150	135	168	180	166	172	142
$20-2-3$	524,3	485	485	681	451	478	506	507	587	531	532
$30-3-4$	730,4	756	756	742	692	707	725	760	713	731	722
$40-3-4$	1301,2	1291	1291	1380	1468	1283	1196	1297	1255	1316	1235
$50-3-6$	1851,3	1778	1782	1844	1986	1650	1820	1750	2047	1981	1875
$70-4-10$	2378,4	2334	2338	2552	2008	2548	2470	2303	2504	2170	2557
$90-4-10$	3691,3	3525	3508	3888	3472	4092	3746	3785	3770	3571	3556
$100-5-12$	3813,2	3809	3800	4111	3718	3855	3874	3637	3633	3442	4253
$120-5-12$	5186,1	5189	5211	5707	4921	5567	5220	4823	5080	4625	5518
$150-6-14$	6233,5	6022	6128	6417	6312	6921	5874	5660	6353	6128	6520
$200-8-20$	7098,1	6799	6929	7397	6667	7204	7211	6732	7434	7304	7304

TABLE 4.6 – Qualité des solutions pour Π_2 ($\min \sum_{i=1}^{n} w_i C_i$) en moins de 120 secondes.

En définitive, quinze paramètres régissent le fonctionnement de la métaheuristique. La gestion de la population de l'algorithme génétique est déterminée par les six paramètres suivants :

N **:** taille de la population.

t_D **:** proportion d'individus générés par croisement.

t_S **:** proportion de ces individus qui restent dans la population.

K **:** nombre d'individus comparés dans la sélection.

n_R **:** nombre d'itérations consécutives avant de réinitialiser la population.

n_A **:** nombre d'itérations consécutives avant de l'algorithme.

L'*intensification*, à savoir l'algorithme de recherche locale, est régi par les neuf paramètres suivants :

ItR **:** nombre d'itérations du recuit simulé.

p_1 **:** probabilité du voisinage de réaffectation.

p_2 **:** probabilité du voisinage de reséquencement.

T **:** température du recuit simulé.

n_p **:** palier de décroissance de la température.

α **:** coefficient de décroissance (géométrique) de la température quand le palier est franchi.

ζ_1 **:** nombre d'itérations consécutives avant redémarrage de la température.

ζ_2 **:** nombre d'itérations consécutives avant diminution de la température de redémarrage.

β **:** coefficient de diminution de la température de redémarrage.

Après une analyse des résultats en fonction des paramètres, plusieurs observations s'imposent :

- Plus la taille de l'instance augmente, plus l'étape d'intensification, associée à l'algorithme de recherche locale, prend de l'importance. C'est pourquoi, avec un seul individu dans la population et environ 30 millions d'itérations, les résultats obtenus sont largement meilleurs qu'avec une population de plusieurs dizaines d'individus.
- Conserver un tel paramétrage sur les instances de petite taille (moins de 40 tâches) empêche de trouver les solutions optimales. C'est là que la *diversification*, associée à l'algorithme génétique, prend tout son sens, permettant, avec une population d'une trentaine d'individus, de converger vers un optimum global.
- Le nombre important de paramètres rend la détermination d'une bonne configuration très délicate. Parfois, plusieurs stratégies différentes mènent à des résultats semblables, mais certaines s'avèrent très mauvaises et il faut les éviter. Généralement, le compromis entre algorithme génétique et recuit simulé est déterminant dans l'obtention de bonnes solutions.

Une poursuite intéressante de ce travail consisterait à construire des formules empiriques pour fixer les paramètres en fonction de l'instance, afin d'améliorer l'utilisation de la méthode. Un bon vecteur de paramètres dépend de la taille de l'instance mais aussi souvent de caractéristiques plus subtiles. On a montré comment utiliser l'algorithme mémétique présenté dans ce chapitre, pour optimiser deux critères différents. Cependant, cette optimisation n'opère que sur un critère à la fois. Il faut donc choisir lequel considérer avant de lancer la résolution. Cet algorithme sera repris dans le cadre d'une étude multicritère et présenté au chapitre 5.

Cet aspect est crucial dans le cadre d'une application industrielle. Même si en général, certains critères sont plus importants, il peut en pratique être préférable de concéder une dégradation d'un critère au profit d'une amélioration importante au sens d'un autre critère. Les réalisations industrielles qui ont émergé de cette thèse ont pour principal objet l'algorithme présenté dans ce chapitre. L'adaptation et les nouveautés liées à la mise en œuvre feront l'objet du chapitre 6.

4.3 Algorithme d'approximation pour la minimisation du nombre de transferts de ressources

Le problème de couverture par des ensembles (*Set Cover*) est un problème NP-difficile au sens fort, généralisant celui du *Vertex Cover*. C'est l'un des pre-

miers problèmes pour lesquels des algorithmes d'approximation ont été proposés et analysés. Il est étudié dès 1974 par Johnson [40] et par Lovász [54]. Johnson [40] propose un algorithme d'approximation de rapport $H(d) = 1 + \frac{1}{2} + \ldots + \frac{1}{d}$ où d est la cardinalité maximale parmi les ensembles couvrants. Ce résultat a été étendu par Chvàtal [15] au cas pondéré du problème (où des poids sont affectés aux ensembles couvrants).

```
1  COUVERTURE-MINIMALE(T, {T_1, ..., T_s})
2  C ← ∅        ▷ Ensemble solution (ensembles couvrants)
3  P ← ∅        ▷ Ensemble des indices des machines
4  tant que T ≠ ∅ faire
5      max ← 1
6      pour i ← 2 à s faire
7          si |T_i| > |T_max| alors
8              max ← i
9      C ← C ∪ T_max        ▷ On garde l'ensemble le plus couvrant
10     P ← P ∪ {max}
11     T ← T \ T_max
12     pour i ← 1 à s faire
13         T_i ← T_i \ T_max        ▷ On retire les éléments déjà couverts
14 retourner (C, P)
```

Algorithme 17: Résolution du problème de couverture minimale

L'algorithme 17 a été proposé par Johnson [40]. C'est un algorithme d'approximation de rapport $H(d)$, qui est la somme partielle d'ordre d de la série harmonique, où d est la cardinalité maximale d'ensembles couvrants. Il consiste simplement, à chaque itération, à inclure dans la solution l'ensemble qui possède le plus d'éléments non couverts. Ceci implique l'utilisation de l'algorithme 18.

Théorème 8 (Minimisation du nombre de déplacements de ressources)**.** L'algorithme 2 est un algorithme d'approximation pour minimiser le nombre de déplacements de ressources, de rapport $1 + \frac{1}{2} + \ldots + \frac{1}{\Delta}$, où Δ est le nombre maximal de tâches nécessitant la même ressource auxiliaire et pouvant être exécutées par une même machine.

Démonstration. Par le rapport d'approximation de l'algorithme glouton de Johnson [40] et la réduction décrite par la proposition 7, on peut établir le fait que

```
1  MIN-MASQUES(Π'_{E_i,M,{A_i}})
2  si |M_{R_i}(i)| ≠ max_{j=0,...,m}{|M_j(i)|} alors
3      M_{R_i}(i) ← M_{R_i}(i) ∪ {J_0}
4      E_i ← E_i ∪ {J_0}
5  ({{T^i_11, ..., T^i_1t_1}, ..., {T^i_s1, ..., T^i_st_s}}, {j_1, ..., j_s}) ←
   COUVERTURE-MINIMALE(E_i, {M_0(i), ..., M_m(i)})
6  g ← 1
7  pour c ← 1 à s faire
8      si j_c = R_i alors
                ▷ La machine où le masque est initialement situé
9          pour k ← 1 à t_c faire
10             si T^i_ck ≠ 0 alors
11                 x^i_g ← j_c
12                 y^i_g ← T^i_ck
13                 g ← g + 1
14 pour c ← 1 à s faire
15     si j_c ≠ R_i alors
16         pour k ← 1 à t_c faire
17             si T^i_ck ≠ 0 alors
18                 x^i_g ← j_c
19                 y^i_g ← T^i_ck
20                 g ← g + 1
21 retourner (x^i_1, ..., x^i_{n_i}, y^i_1, ..., y^i_{n_i})
```

Algorithme 18: Gestion des déplacements de la ressource A_i

l'algorithme 18 permet de résoudre le problème $\Pi'_{E_i,M,\{A_i\}}$ avec un rapport d'approximation $H(\Delta_i)$, où Δ_i est le nombre maximal de tâches de E_i (nécessitant la ressource A_i) pouvant être exécutées par une même machine :

$$\Delta_i = \max_{j=1,\dots,m} \{|Q_j \cap E_i|\}.$$

En effet, il s'agit par définition du cardinal de l'ensemble de cardinalité maximale parmi les $M_j(i)$. Le test de la ligne 2 de l'algorithme MIN-MASQUES permet de maintenir cette propriété : en effet, si la machine M_{R_i} (qui doit obligatoirement figurer dans la solution) possède la plus grande cardinalité, l'algorithme COUVERTURE-MINIMALE l'inclura dans la solution, d'où l'inutilité d'introduire la tâche fictive J_0 et la cardinalité maximale est alors bien Δ_i. Sinon, la cardinalité maximale est supérieure ou égale à $|M_{R_i}(i)| + 1$, et l'introduction de la tâche J_0 n'empêche pas la cardinalité maximale d'être égale à Δ_i.

Notons , pour tout $i \in \{1, \dots, \ell\}$, $(\hat{x}^i, \hat{y}^i)$ les solutions des problèmes $\Pi'_{E_i,M,\{A_i\}}$ fournies par l'algorithme 18. Ainsi, on a les inégalités suivantes, pour tout $i \in \{1, \dots, \ell\}$:

$$\frac{1}{H(\Delta_i)}(f_i(\hat{x}^i, \hat{y}^i) + 1) \leq \min_{(x^i,y^i)\in\chi_1^i} f_i(x^i, y^i) + 1 \leq f_i(\hat{x}^i, \hat{y}^i) + 1$$

D'où :

$$\frac{1}{\max\limits_{i=1,\dots,\ell} H(\Delta_i)} \sum_{i=1}^{\ell}(f_i(\hat{x}^i, \hat{y}^i) + 1) \leq \sum_{i=1}^{\ell} \frac{1}{H(\Delta_i)}(f_i(\hat{x}^i, \hat{y}^i) + 1) \tag{4.3}$$

$$\leq \sum_{i=1}^{\ell} \min_{(x^i,y^i)\in\chi_1^i} f_i(x^i, y^i) + 1 \leq \sum_{i=1}^{\ell} f_i(\hat{x}^i, \hat{y}^i) + 1 \tag{4.4}$$

En posant $\Delta = \max_{i=1,\dots,\ell} \Delta_i$, $(\hat{x}, \hat{y})$ la solution du problème $\Pi'_{J,M,A}$ fournie par l'algorithme 2, on obtient, d'après la proposition 5, les inégalités

$$\frac{1}{H(\Delta)}\ (\tilde{f}(\hat{x}, \hat{y}) + \ell) \leq \min_{(x,y)\in\chi_1} \tilde{f}(x, y) + \ell \leq \tilde{f}(\hat{x}, \hat{y}) + \ell$$

d'où le résultat :

$$\frac{1}{H(\Delta)} \leq \frac{\min_{(x,y)\in\chi_1} \tilde{f}(x, y) + \ell}{\tilde{f}(\hat{x}, \hat{y}) + \ell} \leq 1.$$

□

1 RÉSOLUTION-POP($\Pi_{J,M,A}$)
2 $(x, y) \leftarrow$MIN-DÉPLACEMENTS-MASQUES($\Pi'_{J,M,A}$)
3 **retourner** CONSTRUCTION-ORDONNANCEMENT(x, y)

Algorithme 19: Algorithme de résolution de $\Pi_{J,M,A}$

Analysons à présent la complexité de l'algorithme 19.

Proposition 17. *L'algorithme 19 s'exécute en temps $O(mn)$.*

Démonstration. La ligne 3 de l'algorithme 19 s'exécute en $O(n)$ d'après la proposition 4. La ligne 2, quant à elle, s'exécute en temps C_{dep}, le temps d'exécution de l'algorithme 2. Ainsi, le temps d'exécution est en $C = O(n) + C_{\text{dep}}$. Il nous faut ainsi analyser la complexité de l'algorithme 2.

- La ligne 4 s'exécute en $O(\ell)$ d'après le corollaire 1.
- On a donc $C_{\text{dep}} = O(\ell) + \sum_{i=1}^{\ell} C_{\text{dep}}(i)$, où $C_{\text{dep}}(i)$ est la complexité de l'algorithme 18 pour le problème $\Pi'_{E_i,M,\{A_i\}}$.

On analyse alors la complexité de l'algorithme 18 :

- La ligne 2 s'exécute en $O(m)$ si on suppose connues les cardinalités des ensembles $M_j(i)$.
- La ligne 5 s'exécute en temps $C_{\text{couv}}(i)$, complexité de l'algorithme 17 pour le problème de couverture minimale de l'ensemble E_i par $\{M_0(i), \ldots, M_m(i)\}$.
- Les boucles des lignes 14 à 20 s'exécutent en temps $O(n_i)$.

On en déduit ainsi que :

$$C_{\text{dep}} = O(\ell) + \sum_{i=1}^{\ell} (O(m) + C_{\text{couv}}(i) + O(n_i)) = \tag{4.5}$$

$$O(\ell) + O(n) + \sum_{i=1}^{\ell} (O(m) + C_{\text{couv}}(i)) = O(n) + O(m\ell) + \sum_{i=1}^{\ell} C_{\text{couv}}(i). \tag{4.6}$$

car $\sum_{i=1}^{\ell} n_i = n$ et $\ell \leq n$.

Or, l'algorithme 17 possède une complexité en $O(mn_i)$ d'où le résultat :

$$C = O(n) + O(n) + O(m\ell) + \sum_{i=1}^{\ell} O(mn_i) = O(n) + O(m\ell) + O(mn) = O(mn).$$

□

Dans cette section, un algorithme d'approximation de rapport $H(\Delta) = 1 + \frac{1}{2} + \ldots + \frac{1}{\Delta}$ a été présenté pour résoudre le problème d'ordonnancement avec machines parallèles différentes, contraintes de qualifications des machines, ressources auxiliaires et temps de setup dépendant de la séquence et des machines, pour minimiser le nombre total de déplacements de ressources (de masques). Ici, Δ est le nombre maximal de tâches ayant la même ressource auxiliaire requise et pouvant être exécutées par une même machine. Cet algorithme s'exécute asymptotiquement en temps $O(mn)$. Dans le cas où le nombre de machines n'est pas une constante fixée du problème, on dispose alors d'un algorithme d'approximation pour un problème NP-difficile.

4.4 Conclusion

Un algorithme de type métaheuristique a été présenté pour traiter le problème d'ordonnancement en photolithographie. Il s'agit d'un algorithme mémétique, basé sur une méthode de recherche locale de type *recuit simulé*, ce qui permet de le ranger dans la catégorie des méthodes *hybrides*. Le codage utilisé est d'un intérêt particulier puisqu'il donne la possibilité de représenter un ensemble de solutions dominantes pour les problèmes consistant à maximiser le nombre de plaquettes traitées dans un horizon donné et à minimiser la somme pondérée des dates de fin d'exécution. La méthode de recherche locale, avec ses opérateurs de voisinage, donne une probabilité non nulle d'obtenir une solution optimale. Certaines propriétés du codage utilisé et de l'algorithme, présentées dans ce chapitre, ont permis de développer des résultats intéressants en termes de résolution d'instances de grande taille et d'améliorer notre compréhension du problème. Une avancée dans l'analyse de l'algorithme viserait à établir si la règle de la proposition 16, lorsqu'elle est appliquée, conserve la propriété de connexité des voisinages. En effet, on restreint les possibilités de choix des voisins. La question du bon réglage des paramètres est aussi d'un grand intérêt. Elle permettrait d'exploiter les capacités de l'algorithme de façon optimale.

Un algorithme d'approximation pour minimiser le nombre de transferts de ressources auxiliaires a également été étudié et analysé. Il est issu d'une adaptation d'un algorithme existant pour le problème de couverture par des ensembles. Son rapport d'approximation est égal à la somme partielle d'ordre Δ de la série harmonique, où Δ est le nombre maximal de tâches partageant la même ressource et éligibles sur la même machine.

Le chapitre qui suit permet d'étendre les méthodes décrites jusqu'à présent au cadre multicritère. Cet aspect s'avère extrêmement pertinent à considérer en pratique.

Chapitre 5

Résolution du problème multicritère

Dans les chapitres précédents, le problème d'ordonnancement issu de la photolithographie en fabrication de semi-conducteurs a été décrit. Ici, nous nous intéressons à la minimisation de la somme pondérée des dates de fin d'exécution et la maximisation du nombre de plaquettes produites dans un horizon de temps donné. L'intérêt du travail présenté est la résolution bicritère du problème en considérant ces deux fonctions objectif. Après un exposé bibliographique sur les principaux travaux effectués dans le domaine de l'ordonnancement multicritère, nous présentons une fonction d'agrégation des deux critères que sont la somme pondérée des dates de fin des tâches et le nombre de plaquettes traitees avant un horizon de temps fixé. Un programme linéaire en nombres entiers est présenté pour minimiser cette fonction et apporter des garanties sur la qualité des solutions retournées. Finalement, une adaptation de l'algorithme mémétique présenté au chapitre 4 est présentée pour la résolution de la version tricritère du problème

5.1 État de l'art

Rares sont les problèmes d'optimisation réels pouvant être traités sans considérer plusieurs critères de performance. Ce fait, partagé par les experts de divers domaines industriels, explique l'intérêt toujours croissant pour l'étude des aspects multicritère en optimisation combinatoire. Il est parfaitement envisageable que dans certaines applications où plusieurs objectifs sont importants à prendre en compte, une solution efficace au regard d'un objectif soit mauvaise pour l'autre. On peut à ce titre prendre l'exemple d'une compagnie de transports qui a pour objectifs simultanés la satisfaction de ses usagers et ses propres intérêts économiques. Même si on peut en un sens considérer ces critères comme corrélés, ils traduisent des visions différentes et font apparaître la notion de compromis, essentielle à la compréhension de l'aspect multicritère.

L'optimisation multicritère est un sujet délicat. Là où rechercher des solutions évaluées au moyen de scalaires (généralement appartenant au corps $\mathbb{Q}$ des rationnels) ne nécessite pas d'établir préalablement une relation d'ordre (elle est dans ce cas évidente), le cas multicritère, dans lequel chaque solution est évaluée au

moyen d'un vecteur, pose un premier problème. Il faut définir la relation d'ordre permettant de *discriminer* les solutions entre elles.

Une fois des relations d'ordre établies, on peut déterminer un ensemble d'éléments minimaux au sens de cette évaluation, c'est-à-dire des solutions non dominées. L'énumération de cet ensemble, selon le problème, conduit généralement à une explosion combinatoire (voir par exemple Emelichev et al. [22] sur l'énumération des solutions efficaces pour le problème d'arbre couvrant minimal multicritère). Il convient souvent d'affiner la relation d'ordre en la *prolongeant* en un *ordre total* qui lui est compatible. C'est le principe des *fonctions d'agrégation* ([92], [5], [67] et [28]).

Dans cette section, les principales approches de résolution des problèmes d'ordonnancement multi-critère sont évoquées [82]. On dresse un état de l'art de l'ordonnancement multi-critère, en se concentrant sur les cas proches de la problématique abordée.

5.1.1 Relations d'ordre

En optimisation multicritère, il convient de définir, entre les solutions d'un problème Π, une relation d'ordre qui permet de les discriminer. Soit χ_1 l'ensemble des solutions d'un problème Π et q le nombre de critères considérés. Ainsi, à tout $x \in \chi_1$, on associe l'élément $\phi(x) = (\phi_1(x), \ldots, \phi_q(x)) \in \mathbb{R}^q$, qui est le *vecteur coût*, ou vecteur correspondant à x dans *l'espace des critères* de Π. La i-ème composante de $\phi(x)$ correspond à la valeur de la solution x selon le i-ème critère. Chacun des q critères est soit à maximiser, soit à minimiser.

En l'absence d'informations particulières sur la relation d'ordre qu'un décideur établit entre les solutions, la seule que l'on peut utiliser est la *dominance de Pareto* [71]. Celle-ci se définit de la manière suivante : nous dirons dans la suite qu'une solution x_1 est (resp. *strictement*) *meilleure* sur le i-ème critère qu'une solution x_2 si et seulement si $\phi_i(x_1) \leq \phi_i(x_2)$ (resp. $\phi_i(x_1) < \phi_i(x_2)$) et ce critère est à minimiser, ou $\phi_i(x_1) \geq \phi_i(x_2)$ (resp. $\phi_i(x_1) > \phi_i(x_2)$) et ce critère est à maximiser. Ainsi, on dit que x_1 *domine* (ou est préférée, ou est meilleure que) x_2 au sens de Pareto si et seulement si x_1 est meilleure que x_2 sur tous les critères. On dit aussi, dans ce cas, que $\phi(x_1)$ domine $\phi(x_2)$ au sens de Pareto.

Définition 7. Soient x et y deux solutions d'un problème d'optimisation multicritère. On note $\phi(x) = (\phi_1(x), \ldots, \phi_q(x))$ et $\phi(y) = (\phi_1(y), \ldots, \phi_q(y))$ les évaluations de ces solutions suivant chacun des q critères.

On dit que x *domine* y *au sens de Pareto*, et on note $x \leq_{\mathcal{P}} y$, si et seulement si pour tout i compris entre 1 et q tel que le critère i est à minimiser, $\phi_i(x) \leq \phi_i(y)$,

pour tout i compris entre 1 et q tel que le critère i est à maximiser, $\phi_i(x) \geq \phi_i(y)$ et il existe un indice pour lequel l'inégalité est stricte.

On dit que x *domine strictement* y *au sens de Pareto*, et on note $x <_{\mathcal{P}} y$, si et seulement si pour tout i compris entre 1 et q tel que le critère i est à minimiser, $\phi_i(x) < \phi_i(y)$, pour tout i compris entre 1 et q tel que le critère i est à maximiser, $\phi_i(x) > \phi_i(y)$.

Ainsi, les seules solutions optimales au sens de Pareto sont celles qui ne sont dominées par aucune autre solution au sens de Pareto. Ces définitions permettent donc d'introduire la notion de solution optimale au sens Pareto. En d'autres termes, on considère que les vecteurs sont optimaux si on ne peut améliorer aucune composante sans dégrader la valeur d'une autre composante (voir Pareto [71]).

Définition 8. Soit x une solution d'un problème d'optimisation multicritère. On dit que x est optimale *au sens de Pareto*, ou Pareto-optimale, si et seulement s'il n'existe pas de solution qui domine x au sens de Pareto. On note E l'ensemble des solutions Pareto-optimales. On dit que x est *faiblement* optimale *au sens de Pareto*, ou faiblement Pareto-optimale, si et seulement s'il n'existe pas de solution qui domine strictement x au sens de Pareto. On note F l'ensemble des solutions faiblement Pareto-optimales.

Notons que l'ensemble des solutions optimales au sens de Pareto est inclus dans l'ensemble des solutions faiblement optimales au sens de Pareto : $E \subset F$.

Généralement, il apparaît plus pertinent de rechercher dans E puisque certaines solutions de F présentent souvent peu d'intérêt. Pour une description détaillée des relations d'ordre couramment utilisées dans la littérature en optimisation multicritère, voir Sawaragi et al. [76] et Ehrgott et al. [20].

La génération de l'ensemble des solutions Pareto-optimales a fait l'objet de nombreux travaux sur des problèmes d'optimisation multicritère. En règle générale, ces problèmes deviennent NP-difficiles dans ce contexte. On peut citer Camerini et al. [11] pour l'énumération des solutions Pareto-optimales du problème d'arbres couvrant ou encore Serafini [77] pour l'énumération des solutions Pareto-optimales du problème de plus court chemin. Les problèmes d'ordonnancement font aussi l'objet de ce type d'études ([83]).

5.1.2 Fonctions d'agrégation

Le principe

La relation d'ordre définie par la dominance de Pareto est partielle puisque, par exemple, les solutions x et y telles que $\phi(x) = (1, 18)$ et $\phi(y) = (5, 16)$) (où dans le problème considéré, les deux critères sont à maximiser) ne sont pas comparables au sens de Pareto.

Une alternative serait donc d'établir un ordre total, en se ramenant à l'ordre usuel sur les réels. Ceci par le biais d'une fonction f, appelée *fonction d'agrégation* (ou *scalarisante*) $f : \mathbb{R}^q \to \mathbb{R}$ de sorte que pour tout couple de solutions (x, y), x est préférée à y (au sens de f) si et seulement si $f(\phi(x)) \leq f(\phi(y))$.

Bien entendu, f sera généralement choisie de sorte que pour tout couple de solutions (x, y) tel que x domine y au sens de Pareto, $f(\phi(x)) \leq f(\phi(y))$. Dans ce cas, f est compatible avec la dominance de Pareto.

Cette propriété est posée ci-dessous :

Définition 9 (Fonction d'agrégation croissante, strictement croissante, fortement croissante)**.** Soit $f : \mathbb{R}^q \to \mathbb{R}$ une fonction d'agrégation de q critères. On considère une fonction d'évaluation des critères ϕ associée à un problème d'optimisation.

On dit que f est

- *croissante* avec la dominance de Pareto si et seulement si pour tout couple (x, y) de solutions tel que $x \leq_{\mathcal{P}} y$, on a $f(\phi(x)) \leq f(\phi(y))$.
- *strictement croissante* avec la dominance de Pareto si et seulement si pour tout couple (x, y) de solutions tel que $x <_{\mathcal{P}} y$, on a $f(\phi(x)) < f(\phi(y))$.
- *fortement croissante* avec la dominance de Pareto si et seulement si pour tout couple (x, y) de solutions tel que $x \leq_{\mathcal{P}} y$, on a $f(\phi(x)) < f(\phi(y))$

On peut voir qu'une fonction d'agrégation strictement croissante aura l'intéressante propriété d'avoir comme optimum une solution faiblement Pareto-optimale. Mieux encore, si la fonction d'agrégation est fortement croissante, on sait que la solution retournée sera Pareto-optimale.

On peut ainsi formuler le problème Π_f, consistant à minimiser la fonction d'agrégation f, comme suit :

$$\Pi_f : \min_{x \in \chi_1} f(\phi(x))$$

La somme pondérée est une fonction d'agrégation simple, qui, étant donné un vecteur de poids réels $\lambda = (\lambda_1, \ldots, \lambda_q)$ s'exprime ainsi :

$$f_\lambda : \mathbb{R}^q \to \mathbb{R}$$
$$(x_1, \ldots, x_q) \mapsto f_\lambda(x_1, \ldots, x_q) = \sum_{i=1}^{q} \lambda_i x_i.$$

Cette fonction d'agrégation est simple à mettre en œuvre et toute solution optimale au sens de la somme pondérée fait partie de l'ensemble des solutions faiblement Pareto-optimales. De plus, son paramétrage est relativement aisé. Il faut veiller à adapter le signe du coefficient λ_i, qui, lorsqu'on cherche à minimiser la fonction, doit être positif si le critère i est à minimiser et négatif s'il est à maximiser. Cependant, elle ne permet d'atteindre que les points extrêmes de l'enveloppe convexe des solutions du problème. Ce sont les solutions dites *supportées*. Or, certaines solutions Pareto-optimales n'appartiennent pas à cet ensemble. Il y a inclusion stricte dans le cas général. Bien que dans certains problèmes à la structure particulière, ces ensembles coïncident (voir par exemple White [88] qui montre en 1984 que pour le problème d'affectation multi-agent, toutes les solutions Pareto-optimales sont supportées), ceci est une limitation importante de cette fonction d'agrégation.

L'*ordre lexicographique* est une autre façon naturelle d'établir une relation d'ordre totale en optimisation multicritère (voir [8]). On définit cette relation, notée Lex, de la manière suivante : $x <_{\text{Lex}} y$ si et seulement s'il existe k tel que $1 \leq k \leq q$, $\phi_k(x) < \phi_k(y)$, et pour tout i entre 1 et $k-1$, $\phi_i(x) = \phi_i(y)$. Notons qu'on peut définir un ordre lexicographique différent en effectuant une permutation des q critères considérés. On obtient ainsi une relation d'ordre total et elle est compatible avec la dominance de Pareto. En effet, si $x <_{\mathcal{P}} y$, alors $x <_{\text{Lex}} y$.

La norme de Tchebychev

Plusieurs fonctions d'agrégation ont été étudiées dans la littérature sur l'optimisation multi-critère. On peut entre autres citer la somme pondérée, les *Ordered Weighted Average* (ou moyenne ordonnée pondérée) (voir [92]), l'intégrale de Choquet (voir [14]), les distances à un point de référence (voir [89]), etc.

Nous décrirons ici une fonction d'agrégation dont certaines propriétés nous permettent par la suite d'utiliser des résultats déjà établis sur le problème d'ordonnancement étudié. Il s'agit d'une fonction de la catégorie des distances à un point de référence, beaucoup étudiées dans la littérature (voir par exemple [89], [69] et [56]) : la *norme de Tchebychev pondérée*.

La norme de Tchebychev est une fonction d'agrégation qui évalue une solution en fonction de sa distance à un *point de référence* selon la norme infinie L_∞. Étant

donnés un jeu de poids $\lambda = (\lambda_1, \ldots, \lambda_q)$ et un point de référence $r = (r_1, \ldots, r_q) \in \mathbb{R}^q$, elle est définie ainsi :

$$\begin{aligned} s_{\lambda,r} : \mathbb{R}^q &\to \mathbb{R} \\ (x_1, \ldots, x_q) &\mapsto s_{\lambda,r}(x_1, \ldots, x_q) = \|(\lambda_1(x_1 - r_1), \ldots, \lambda_q(x_q - r_q))\|_\infty . \end{aligned}$$

Lorsque le point de référence r est bien choisi, pour toute solution x optimale au sens de Pareto, il existe un vecteur $\lambda \in \mathbb{R}^q$ tel que x est une solution optimale au sens de $s_{\lambda,r}$. Ce résultat est dû à Wierzbicki [90]. Notons que généralement, le point de référence est déterminé par le décideur et représente ses aspirations sur les valeurs des différents critères.

Ainsi, toute solution optimale au sens de Pareto peut être obtenue par la recherche d'une solution optimale au sens de la norme de Tchebychev pondérée. Le point de référence et le jeu de poids sont des données déterminantes dans cette recherche.

Afin de donner une condition suffisante sur ces données pour que la solution optimale au sens de $s_{\lambda,r}$ soit optimale au sens de Pareto, on définit le *point idéal* $x^* \in \mathbb{R}^q$ dont toutes les composantes sont les valeurs optimales des critères correspondants.

Définition 10. Soit Π un problème d'optimisation à q critères. Le vecteur $x^* = (x_1^*, \ldots, x_q^*)$ est appelé *point idéal* si et seulement si pour tout indice i compris entre 1 et q, x_i^* est la valeur de la solution optimale selon le i-ème critère.

Wierzbicki [90] montre en 1986 que si le point de référence choisi r domine le (ou est égal au) point idéal x^* au sens de Pareto, alors pour toute solution x_1 optimale au sens de Pareto, il existe un jeu de poids strictement positifs λ tel que $x_1 \in \mathrm{Argmin}_{x \in \chi_1} s_{\lambda,r}(x)$. De plus, quels que soient les poids, la fonction est strictement croissante si le point de référence domine le (ou est égal au) point idéal au sens de Pareto. Cela signifie que toute solution retournée est faiblement Pareto-optimale. Par ailleurs, parmi les solutions retournées, l'une d'elles est Pareto-optimale. Ce qui implique que si une unique solution est retournée, elle est Pareto-optimale.

Cette propriété fait tout l'intérêt de la norme de Tchebychev dans la résolution d'un problème multicritère dans la mesure où toute solution optimale au sens de Pareto est susceptible d'etre calculée, ce qui n'est pas le cas pour une fonction d'agrégation telle que la somme pondérée. En effet, il existe généralement des solutions optimales au sens de Pareto qu'aucune combinaison linéaire des valeurs des critères ne permet d'obtenir par minimisation ou maximisation. Ces solutions optimales sont dites *non supportées*.

Ainsi, on dispose d'une fonction d'agrégation (norme de Tchebychev pondérée) qui permet de calculer des solutions non supportées optimales au sens de Pareto. D'autres fonctions d'agrégation avec les mêmes propriétés ont été proposées.

Ruiz et al. [56] ont proposé la fonction : $f(x_1, \ldots, x_q) = \sum_{i=1}^{q} \max(0, \lambda_i(x_i - r_i))$.

Miettinen et al. [69] ont proposé une généralisation, avec un paramètre p :

$$f_p(x_1, \ldots, x_q) = \max_{I \subset [1,q], |I|=p} \sum_{i \in I} \max(0, \lambda_i(x_i - r_i)).$$

On voit que $f_q = f$.

Une autre approche, proposée par Gembicki [26], permet de s'affranchir de la détermination d'un point de référence qui domine le point idéal. On considère un point de référence quelconque r et un vecteur $\lambda = (\lambda_1, \ldots, \lambda_q)$ de poids positifs. La fonction d'agrégation

$$G_{\lambda,r} : \mathbb{R}^q \to \mathbb{R}$$
$$(x_1, \ldots, x_q) \mapsto G_{\lambda,r}(x_1, \ldots, x_q) = \min_{1 \leq i \leq q} \left\{ \frac{1}{\lambda_i}(x_i - r_i) \right\}$$

a pour maxima des solutions faiblement Pareto-optimales. Cependant, le choix des poids est délicat, puisque pour un jeu de poids fixé, la solution (faiblement Pareto-optimale) obtenue dépend fortement de la position du point de référence choisi.

Avant de dresser un état de l'art succinct des travaux menés en ordonnancement multicritère, on présente l'approche ε-contrainte, proposée par Haimes et al. [33]. On considère un problème d'optimisation à q critères. Soit k tel que $1 \leq k \leq q$. Supposons qu'il existe un vecteur $\varepsilon_k = (\varepsilon_k^1, \ldots, \varepsilon_k^{k-1}, \varepsilon_k^{k+1}, \ldots, \varepsilon_k^q)$ tel que le problème monocritère consistant à optimiser le critère k, auquel on a rajouté les contraintes $\phi_i(x) \leq \varepsilon_k^i$ pour tout critère i à minimiser et $\phi_j(x) \geq \varepsilon_k^j$ pour tout critère j à maximiser, admet une solution. Alors cette solution est faiblement Pareto-optimale.

5.1.3 Ordonnancement multicritère

S'agissant de problèmes d'ordonnancement, la plupart des travaux existants (par exemple Della Croce et al. [18] et Estève et al. [24]) traitent du cas à une machine. Smith [79] traîte par exemple du problème à une machine avec l'approche ε-contrainte bicritère tenant compte de la somme pondérée des dates de fin d'exécution et du retard maximal des tâches. Emmons [23] montre que pour l'ordre lexicographique, tout problème d'ordonnancement bicritère à une machine tenant

compte de la somme des dates de fin et d'un autre critère régulier, appartient à la classe P. John [39] arrive à borner le nombre de solutions efficaces par une fonction exponentielle de la taille du problème, puisqu'elle dépend de $p_{max} - p_{min}$, l'écart entre la plus longue et la plus courte durée d'exécution.

Lorsque des poids sont affectés aux tâches, la somme pondérée des dates de fin, toujours dans le cas de problèmes à une machine, a été étudiée avec d'autres critères. Pour l'ordre lexicographique bicritère, Hoogeveen [37] montre qu'avec le retard maximal, le problème est NP-difficile. Chand et al. [12] le résolvent à l'aide d'un algorithme de programmation dynamique. L'approche ε-contrainte fixant le retard maximal à 0 est NP-difficile au sens fort. Les mêmes auteurs [13] étudient cette version. Miyazaki [62] propose une méthode efficace de résolution pour ce problème. Les méthodes les plus utilisées pour ces problèmes sont à base de branch-and-bound (méthodes par séparation et évaluation) et de programmation dynamique.

Les problèmes d'ordonnancement à machines parallèles sont des généralisations de problèmes à une machine, et sont en ce sens plus difficiles à résoudre. Mais ils sont plus importants en pratique, puisque plusieurs ressources sont souvent disponibles pour réaliser des tâches dans les systèmes industriels. Le problème $P|pmtn|\,\mathrm{Lex}(\sum_j C_j, C_{\max})$ à machines parallèles avec préemption des tâches, et ordre lexicographique sur la somme des dates de fin des tâches et le makespan, est polynomial. Leung et Young [52] proposent un algorithme de complexité $O(n \log n)$ pour le résoudre. McCormick et Pinedo [57] considèrent le cas $Q|pmtn|\varepsilon(\sum_j C_j, C_{\max})$ avec l'approche ε-contrainte, et proposent un algorithme polynomial qui génère l'ensemble des solutions strictement Pareto-optimales.

Les problèmes d'ordonnancement à machines parallèles sont très peu traités avec des fonctions d'agrégation de la catégorie des distances à un point de référence. Nikulin et al. [69], ainsi que Luque [55], étudient ce type de méthodes et proposent plusieurs variantes de la norme de Tchebychev pondérée augmentée. Les propriétés de cette fonction justifient la formulation proposée dans la suite.

5.2 Méthode exacte avec fonction d'agrégation

Soit Π_1 (resp. Π_2) le problème consistant à maximiser le nombre de plaquettes produites avant H (resp. minimiser la somme pondérée des dates de fin d'exécution). Les problèmes Π_1 et Π_2 ont été définis au chapitre 1, section 1.2 et résolus dans les chapitres 3 et 4. On note χ l'ensemble des solutions réalisables de Π_1 et Π_2.

5.2.1 Formalisation du problème

La norme de Tchebychev pondérée possède des propriétés adaptées à une résolution exacte du problème d'ordonnancement qui nous occupe dans ce document. Facile à exprimer dans un programme linéaire en nombre entiers, elle permet d'atteindre tout point de l'ensemble des solutions Pareto-optimales, et n'admet pour minimum que des solutions faiblement Pareto-optimales.

On peut maintenant définir le problème POP-BCT (problème d'ordonnancement issu de la photolithographie, bicritère, minimisation de la norme de Tchebychev pondérée), consistant à minimiser la norme de Tchebychev pondérée, par rapport à un point de référence (dominant le point idéal au sens de Pareto), pour les deux critères qui nous intéressent, à savoir le nombre de plaquettes produites avant un horizon donné, à maximiser (qui sera le premier critère), et la somme pondérée des dates de fin d'exécution, à minimiser (le deuxième critère). Dans ce qui suit, on notera Π_B ce problème.

Le problème bicritère POP-BCT

Soit ϕ_1 (resp. ϕ_2) l'application qui à toute solution $x \in \chi$ associe son coût selon le premier (resp. le deuxième) critère.

On a

$$\Pi_B : \min_{x \in \chi} s_{\lambda,r}(\phi_1(x), \phi_2(x))$$

où $\lambda \in \mathbb{R}^2$ et $r \in \mathbb{R}^2$.

On peut voir que ce problème est NP-difficile au sens fort.

Théorème 9. Le problème Π_B est NP-difficile au sens fort.

Démonstration. En remarquant que pour $\lambda_2 = 0$ et $\lambda_1 = 1$, avec r_1 supérieur à la valeur de la solution optimale selon le premier critère, on se ramène à la maximisation du nombre de plaquettes produites avant un horizon donné, problème qui est NP-difficle au sens fort, selon le théorème 2. □

Nous nous bornerons donc à utiliser un point r dominant le point idéal au sens de Pareto. La proposition suivante donne un point de référence vérifiant une telle propriété.

Proposition 18. *Soit $I_{\lambda,r}$ une instance du problème Π_B avec comme fonction objectif $s_{\lambda,r}$. Avec*

$$r = (\sum_{i=1}^{n} c_i, \sum_{i=1}^{n} w_i(\min_{j \in \mathcal{M}_i} \rho_{ij})),$$

pour toute solution $x \in \chi$ optimale au sens de Pareto, il existe $\lambda' \in (\mathbb{R}_+^*)^2$ tel que x est une solution optimale au sens de $s_{\lambda',r}$.

De plus, pour tout $\lambda' \in (\mathbb{R}_+^*)^2$, toute solution optimale au sens de $s_{\lambda',r}$ est faiblement Pareto-optimale.

Démonstration. Il suffit de montrer que pour toute instance I de Π_B, r domine le point idéal au sens de Pareto, ce qui permet de conclure, par le résultat de Wierzbicki, évoqué en section 5.1.2. Pour toute solution $x \in \chi$, on a $\phi_1(x) = \sum_{i=1}^{n} c_i\theta_i$ avec $0 \leq \theta_i \leq 1$ pour tout $i \in \{1, \ldots, n\}$. On en déduit que

$$\phi_1(x) \leq \sum_{i=1}^{n} c_i$$

donc la valeur de la première composante du point idéal est inférieure ou égale à la valeur de la première composante de r, ce qui correspond à un critère à maximiser. De plus, en notant C_i la date de fin d'exécution de J_i, on a

$$\phi_2(x) = \sum_{i=1}^{n} w_i C_i \leq \sum_{i=1}^{n} w_i (\min_{j \in \mathcal{M}_i} \rho_{ij}).$$

D'où le résultat. □

Formulation PLNE pour POP-BCT

On peut utiliser les résultats évoqués pour le problème POP afin d'obtenir une formulation pour la version bicritère avec minimisation de la norme de Tchebychev.

Le programme linéaire mixte (FBCT1) suivant est valide pour le problème Π_B ayant pour fonction objectif $s_{\lambda,r}$, avec $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}_+^2$ et $r = (r_1, r_2) \in \mathbb{R}^2$, tel

que r domine le point idéal de Π_B au sens de Pareto.

$$\text{Min } z \tag{5.1}$$

$$\text{s. c.}$$

$$z \geq \lambda_1 \left(r_1 - \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right) \right) \tag{5.2}$$

$$z \geq \lambda_2 \left(\sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right) - r_2 \right) \tag{5.3}$$

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1 \qquad \forall i = 1, \ldots, n \tag{5.4}$$

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0})}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0}+1)}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i \tag{5.5}$$

$$u_{ijt} + \sum_{t_0=\max(0,t-\rho_{i_0 j}+1-\beta_{ij i_0})}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{ij i_0})} u_{i_0 j t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\} \tag{5.6}$$

$$\sum_{i=1}^{n} \sum_{\substack{j \in \mathcal{M}_i \\ j \neq R_{\varphi_i}}} u_{ij0} = 0 \tag{5.7}$$

$$u_{ijt} \in \{0,1\} \qquad \forall i = 1, \ldots, n, j \in \mathcal{M}_i, t = 0, \ldots, T-\rho_{ij}. \tag{5.8}$$

Les contraintes (5.4) à (5.7), présentées à la section 3.2.1 du chapitre 3, caractérisent l'ensemble χ des solutions réalisables du problème.

Les contraintes (5.2) et (5.3) garantissent le fait que

$$z \geq \max(\lambda_1 |\phi_1(x) - r_1|, \lambda_2 |\phi_2(x) - r_2|)$$

pour tout $x \in \chi$. Comme la solution optimale correspond à la valeur minimale vérifiant cette condition, on a pour toute solution optimale x de (FBCT1), l'égalité

$$z = \max(\lambda_1 |\phi_1(x) - r_1|, \lambda_2 |\phi_2(x) - r_2|) = \max(\lambda_1 (r_1 - \phi_1(x)), \lambda_2 (\phi_2(x) - r_2)).$$

On remarque que la dernière égalité est déduite du fait que le premier critère est à maximiser, le second est à minimiser, et que r domine le point idéal au sens de

Pareto. z est aussi toujours positive, puisque λ_1 et λ_2 sont supposés positifs (cela suffit pour pouvoir trouver les solutions optimales au sens de Pareto).

Cette égalité correspond donc bien au fait que la solution optimale du programme linéaire mixte est la valeur $\min_{x \in \chi} s_{\lambda,r}(x)$. Il s'agit là d'une linéarisation classique du maximum d'un ensemble de valeurs que l'on peut exprimer linéairement en fonction des variables de décision.

On obtient ainsi un moyen, étant données quatre valeurs $\lambda_1 > 0$, $\lambda_2 > 0$, r_1 et r_2, de calculer la solution $x \in \chi$ qui minimise la norme de Tchebychev pondérée par rapport à un point de référence. La proposition 18 suggère un point de référence facile à calculer qui satisfait à la condition suffisante pour qu'il existe un jeu de poids qui permette d'obtenir une solution optimale au sens de Pareto. De plus, avec un tel point, toute solution retournée possède la garantie d'être faiblement Pareto-optimale, quels que soient les poids choisis. Pour finir, on propose une formulation renforcée du problème.

Le programme linéaire mixte (FBCT1-RF) est valide pour le problème Π_B ayant pour fonction objectif $s_{\lambda,r}$, avec $\lambda = (\lambda_1, \lambda_2) \in \mathbb{R}^2_+$ et $r = (r_1, r_2) \in \mathbb{R}^2$ tel que r domine le point idéal de Π_B au sens de Pareto.

$$\text{Min } z \tag{5.9}$$

s. c.

$$z \geq \lambda_1 \left(r_1 - \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right) \right) \tag{5.10}$$

$$z \geq \lambda_2 \left(\sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right) - r_2 \right) \tag{5.11}$$

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1 \qquad \forall i = 1, \ldots, n \tag{5.12}$$

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0})}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0}+1)}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i \tag{5.13}$$

$$u_{ijt} + \sum_{t_0=\max(0,t-\rho_{i_0 j}+1-\beta_{ji i_0})}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{ij i_0})} u_{i_0 j t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\} \tag{5.14}$$

$$\sum_{i=1}^{n} \sum_{\substack{j \in \mathcal{M}_i \\ j \neq R_{\varphi_i}}} u_{ij0} = 0 \tag{5.15}$$

$$u_{ijt} \in \{0,1\} \qquad \forall i = 1, \ldots, n, j \in \mathcal{M}_i, t = 0, \ldots, T-\rho_{ij}. \tag{5.16}$$

On l'écrit à partir de la formulation (FBCT1) et de la formulation renforcée des problèmes Π_1 et Π_2 déjà présentée à la section 3.2.4 du chapitre 3.

5.2.2 Formulation étendue

Dans cette section, nous étendons le modèle présenté à une fonction d'agrégation fortement croissante, ce qui permet de garantir que les solutions optimales retournées sont Pareto-optimales et non plus seulement faiblement Pareto-optimales.

Introduisons cette fonction à l'aide d'un exemple. Soient x_a et x_b deux solutions d'une instance I du problème Π_B de vecteurs de coûts respectifs (a_1, a_2) et (b_1, b_2). Supposons que $a_1 = b_1$ et $a_2 < b_2$. On peut en déduire que $x_a \leq_{\mathcal{P}} x_b$. De plus, soit $r = (r_1, r_2)$ un point de référence dominant le point idéal dans l'instance I. Si la fonction d'agrégation est la norme de Tchebychev pondérée avec (r_1, r_2) comme

point de référence et un vecteur $\lambda = (\lambda_1, \lambda_2)$ de poids strictement positifs, on a : $s_{\lambda,r}(x_a) = s_{\lambda,r}(x_b)$ si $\lambda_1(r_1 - a_1) > \max(\lambda_2(a_2 - r_2), \lambda_2(b_2 - r_2)) = \lambda_2(b_2 - r_2)$.

Cet exemple illustre bien un cas où une solution faiblement Pareto-optimale pourrait minimiser $s_{\lambda,r}$. C'est bien une fonction d'agrégation strictement croissante.

Considérons à présent la fonction

$$s^A_{\lambda,r,\varepsilon}(x_1, x_2) = \max(\lambda_1(r_1 - x_1), \lambda_2(x_2 - r_2)) + \varepsilon\left(\lambda_1(r_1 - x_1) + \lambda_2(x_2 - r_2)\right),$$

où $\varepsilon > 0$.

Ici, on voit bien que, si $\lambda_1(r_1 - a_1) > \lambda_2(b_2 - r_2)$, alors

$$\begin{aligned} s^A_{\lambda,r,\varepsilon}(a_1, a_2) &= s_{\lambda,r}(a_1, a_2) + \varepsilon\left(\lambda_1(r_1 - a_1) + \lambda_2(a_2 - r_2)\right) \\ &= s_{\lambda,r}(b_1, b_2) + \varepsilon\left(\lambda_1(r_1 - a_1) + \lambda_2(a_2 - r_2)\right) \\ &< s_{\lambda,r}(b_1, b_2) + \varepsilon\left(\lambda_1(r_1 - b_1) + \lambda_2(b_2 - r_2)\right) \\ &= s^A_{\lambda,r,\varepsilon}(b_1, b_2). \end{aligned}$$

Ainsi, les points (a_1, a_2) et (b_1, b_2) ont des images différentes par cette fonction, même s'ils sont égaux sur l'un des critères. Un résultat fondamental concernant la fonction $s^A_{\lambda,r,\varepsilon}$ est qu'elle est fortement croissante.

La *norme de Tchebychev pondérée augmentée* selon les paramètres

$$\lambda = (\lambda_1, \ldots, \lambda_q), r = (r_1, \ldots, r_q)$$

et $\varepsilon \in \mathbb{R}^*_+$ est la fonction

$$\begin{aligned} s^A_{\lambda,r,\varepsilon} : \mathbb{R}^q &\to \mathbb{R} \\ (x_1, \ldots, x_q) &\mapsto \|(\lambda_1|x_1 - r_1|, \ldots, \lambda_q|x_q - r_q|)\|_\infty + \varepsilon \sum_{k=1}^{q} \lambda_k|x_k - r_k|. \end{aligned}$$

La norme de Tchebychev pondérée augmentée est une fonction d'agrégation fortement croissante pour un problème si $\varepsilon > 0$, si les poids sont positifs et si le point de référence domine le point idéal du problème.

Pour des paramètres $(\lambda, r, \varepsilon)$ fixés, les solutions optimales du programme linéaire mixte (FBCT-RF) suivant sont des solutions Pareto-optimales pour le problème d'ordonnancement en photolithographie avec le nombre de plaquettes produites avant l'horizon H, à maximiser, et la somme pondérée des dates de fin d'exécution, à minimiser, comme critères, lorsque (r_1, r_2) domine le point idéal et $\varepsilon, \lambda_1, \lambda_2 > 0$.

$$\text{Min } z + \varepsilon\lambda_1 \left(r_1 - \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right) \right) \tag{5.17}$$
$$+\varepsilon\lambda_2 \left(\sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right) - r_2 \right)$$

s. c.

$$z \geq \lambda_1 \left(r_1 - \sum_{i=1}^{n} \sum_{j \in \mathcal{M}_i} \left(\sum_{t=0}^{H-\rho_{ij}} c_i u_{ijt} + \sum_{t=H-\rho_{ij}+1}^{H} c_i \frac{H-t}{\rho_{ij}} u_{ijt} \right) \right) \tag{5.18}$$

$$z \geq \lambda_2 \left(\sum_{i=1}^{n} w_i \left(\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt}(t+\rho_{ij}) \right) - r_2 \right) \tag{5.19}$$

$$\sum_{j \in \mathcal{M}_i} \sum_{t=0}^{T-\rho_{ij}} u_{ijt} = 1 \qquad \forall i = 1, \ldots, n \tag{5.20}$$

$$u_{ijt} + \sum_{j_0 \in \mathcal{M}_{i_0} \setminus \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0})}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij})} u_{i_0 j_0 t_0} + \sum_{j_0 \in \mathcal{M}_{i_0} \cap \{j\}} \sum_{t_0=\max(0,t-\rho_{i_0 j_0}+1)}^{\min(T-\rho_{i_0 j_0}, t+\rho_{ij}-1)} u_{i_0 j_0 t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \neq i \text{ tel que } \varphi_{i_0} = \varphi_i \tag{5.21}$$

$$u_{ijt} + \sum_{t_0=\max(0,t-\rho_{i_0 j}+1-\beta_{i_0 ji})}^{\min(T-\rho_{i_0 j}, t+\rho_{ij}-1+\beta_{ij i_0})} u_{i_0 j t_0} \leq 1$$
$$\forall i = 1, \ldots, n, \forall j \in \mathcal{M}_i, \forall t = 0, \ldots, T-\rho_{ij}, \forall i_0 \in Q_j \setminus \{i\} \tag{5.22}$$

$$\sum_{i=1}^{n} \sum_{\substack{j \in \mathcal{M}_i \\ j \neq R_{\varphi_i}}} u_{ij0} = 0 \tag{5.23}$$

$$u_{ijt} \in \{0,1\} \qquad \forall i = 1, \ldots, n, j \in \mathcal{M}_i, t = 0, \ldots, T-\rho_{ij}. \tag{5.24}$$

On reconnaît les contraintes (5.20) à (5.23) du problème d'ordonnancement en photolithographie et les contraintes (5.18) et (5.19) qui définissent z comme

$$\max(\lambda_1 |\phi_1(x) - r_1|, \lambda_2 |\phi_2(x) - r_2|).$$

L'expression à minimiser est donc bien la norme de Tchebychev pondérée augmentée.

La figure 5.1 représente un exemple de solutions d'une instance à 8 tâches, 2 machines et 3 masques dans l'espace des critères. On observe, sur le graphe en abscisses, la valeur selon le critère du nombre de plaquettes avant H et en ordonnées,

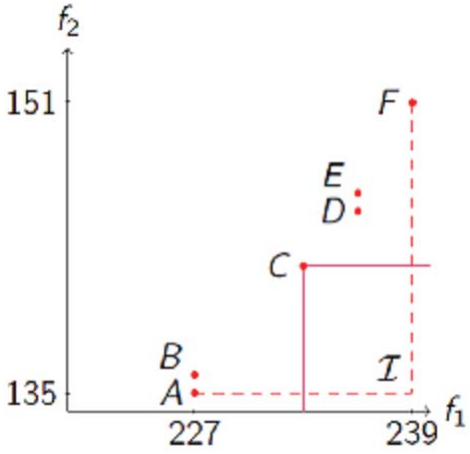

FIGURE 5.1 – Génération de solutions Pareto-optimales pour une instance à 8 tâches, 2 machines et 3 masques

la somme pondérée des dates de fin des tâches. Pour plusieurs instanciations des paramètres du vecteur λ, différentes solutions sont déterminées et représentées par les points affichés sur la figure. Ici, une instance à 8 tâches, 2 machines et 3 masques est résolue au moyen du programme linéaire (FBCT-RF). Le point A correspond à la solution optimale obtenue pour $(\lambda_1, \lambda_2) = (0, 1)$. Le point F correspond à $(\lambda_1, \lambda_2) = (1, 0)$. Ces deux solutions sont en réalité obtenues par une optimisation mono-critère. Cela permet de fixer un point idéal $\mathcal{I}$ et lancer la méthode exacte à base de programmation linéaire pour des vecteurs λ à composantes non nulles. Le point E correspond à $(0,8; 0,2)$ tandis que D est obtenu pour $(0,7; 0,3)$. Avec $(0,5; 0,5)$, on obtient le point C. Finalement, le point B est l'optimum pour les coefficients $(0,1; 0,9)$. Remarquons que la fonction d'agrégation utilisée ici est la norme de Tchebychev pondérée. La norme de Tchebychev pondérée augmentée aurait permis d'éviter les points faiblement Pareto-optimaux que sont B et E.

5.2.3 Calcul des paramètres des formulations

On sait que les coordonnées du point de référence (r_1, r_2) permettent de garantir la Pareto-optimalité des solutions lorsque (r_1, r_2) domine le point idéal. Dans une méthode de résolution, on les détermine alors en calculant les bornes de relaxation linéaire par les formulations monocritère. S'agissant de bornes de relaxation, le point idéal est bien dominé par (r_1, r_2).

Fixer les valeurs des poids des critères est une étape qui se heurte à une difficulté particulière : les deux critères n'ont pas la même unité (un nombre de plaquettes

et une durée pondérée). Pour déterminer les coefficients λ_1, λ_2 de sorte à ramener les écarts sur les critères à la même échelle, on définit le *point anti-idéal*, appelé aussi *point Nadir*.

Définition 11 (Point Nadir)**.** Soit P l'ensemble des solutions Pareto-optimales d'un problème d'optimisation à q critères et pour tout i entre 1 et q, $\phi_i(x)$ l'évaluation de la solution x selon le i-ème critère. Le point anti-idéal, ou point Nadir, noté $N = (x_1^N, \ldots, x_q^N)$ est défini ainsi : pour tout i tel que $1 \leq i \leq q$, si le critère i est à minimiser (resp. à maximiser), alors $x_i^N = \text{Argmax}_{x \in P}\, \phi_q(x)$ (resp. $\text{Argmin}_{x \in P}\, \phi_q(x)$). Il s'agit du point formé des pires coûts sur chaque critère parmi les solutions Pareto-optimales.

Il existe plusieurs façons de déterminer ce point (pour une description de diverses méthodes déjà utilisées, voir Miettinen et al. [41]). Considérons le point Nadir $N = (x_1^N, x_2^N)$ pour une instance de POP-BCT. On note c le coefficient qui indique l'importance relative des deux critères. En effet, comme on peut, selon les cas, privilégier un critère par rapport à l'autre, on multiplie le facteur de normalisation f du critère à privilégier par c, où c est un réel supérieur à 1 dont la valeur indique le degré de prépondérance du critère. Le facteur de normalisation d'un critère est calculé à partir de N comme suit :

$$f = \frac{r_1 - x_1^N}{x_2^N - r_2}.$$

Et on pose alors :

$$\lambda_1 = 1, \qquad \lambda_2 = c \times \frac{r_1 - x_1^N}{x_2^N - r_2}.$$

En effet, puisqu'il s'agit de comparer deux valeurs sur la même échelle, on peut poser $\lambda_1 = 1$ et multiplier par $r_1 - x_1^N$ dans λ_2. Il reste donc à déterminer le point Nadir. Dans la suite, on montre comment déterminer une approximation du point Nadir.

Soient x_1^* une solution optimale selon le critère 1 et x_2^* une solution optimale selon le critère 2. Alors le point $(\phi_1(x_2^*), \phi_2(x_1^*))$ est une approximation du point Nadir. En effet, supposons qu'il existe une solution $x \in P$ (P étant l'ensemble des solutions Pareto-optimales) telle que $\phi_2(x) > \phi_2(x_1^*)$. Alors

$$(\phi_1(x_1^*), \phi_2(x_1^*)) \leq_{\mathcal{P}} (\phi_1(x), \phi_2(x))$$

d'où la contradiction.

De même, supposons qu'il existe une solution $y \in P$ telle que $\phi_1(y) < \phi_1(x_2^*)$. Le point $(\phi_1(x_2^*), \phi_2(x_2^*))$ domine le point $(\phi_1(y), \phi_2(y))$ au sens de Pareto, d'où la contradiction.

On déduit de ce qui précède qu'aucune solution Pareto-optimale n'a une valeur selon le critère i qui soit plus mauvaise que la i-ème coordonnée du point $(\phi_1(x_2^*), \phi_2(x_1^*))$.

Remarque 9. Si on remplace x_1^* et x_2^* par les meilleures solutions connues, l'approximation sera d'autant meilleure que ces solutions sont proches de l'optimum.

Remarque 10. Le cas où $r_1 = x_1^N$ correspond au cas où le point idéal peut être atteint, si x_1^N est calculé comme décrit ci-dessus. De même lorsque $r_2 = x_2^N$. Ainsi, dans ces cas-là, inutile d'utiliser les pondérations, puisqu'une seule solution, déjà trouvée, peut être retournée : celle qui permet d'atteindre le point idéal.

Le paramètre ε utilisé dans la norme pondérée augmentée, intervient pour discriminer les solutions faiblement Pareto-optimales. Le terme linéaire de la valeur $s_{\lambda,r,\varepsilon}(x)$, égal à $\varepsilon \sum_{i=1}^{2} \lambda_i |x_i - r_i|$ doit être inférieur au terme correspondant à la norme de Tchebychev pondérée : $\max_{i=1,2}\{\lambda_i |x_i - r_i|\}$. La raison en est que ce terme ne doit intervenir que lorsqu'il y a égalité au sens de la norme classique de Tchebychev pondérée. On peut alors prendre ε tel que $\varepsilon \sum_{i=1}^{2} \lambda_i |x_i - r_i| < \max_{i=1,2}\{\lambda_i |x_i - r_i|\}$, soit encore

$$\varepsilon < \frac{\max_{i=1,2}\{\lambda_i |x_i - r_i|\}}{\sum_{i=1}^{2} \lambda_i |x_i - r_i|} < 1.$$

Comme $\frac{\max_{i=1,2}\{\lambda_i |x_i - r_i|\}}{\sum_{i=1}^{2} \lambda_i |x_i - r_i|} > \frac{\max_{i=1,2}\{\lambda_i |x_i - r_i|\}}{2 \max_{i=1,2}\{\lambda_i |x_i - r_i|\}} = \frac{1}{2}$, on prend $0 < \varepsilon < \frac{1}{2}$. En général, on prend ε proche de zéro, par exemple 10^{-k} avec k entier suffisamment grand.

5.2.4 Conclusion

Dans cette section, une formulation en programmation linéaire en variables mixtes a été proposée pour la résolution d'un problème d'optimisation bicritère, dont la fonction objectif est une fonction d'agrégation de deux des critères étudiés : la norme de Tchebychev pondérée par rapport à un point de référence. Cette fonction est intéressante dans la mesure où, bien paramétrée, elle permet d'atteindre n'importe quelle solution optimale au sens de Pareto. Cette propriété appréciable n'est par exemple pas garantie dans le cas de la somme pondérée, pour laquelle il existe des solutions (dites solutions non supportées) qui sont optimales au sens de Pareto et qui ne peuvent être obtenues quels que soient les coefficients de la somme pondérée.

Cette méthode suppose la détermination préalable d'un point de référence acceptable, c'est-à-dire pour chaque critère, un majorant (ou un minorant) de la

valeur optimale. On évoque une manière de déterminer ce point en pratique. Il se calcule en fonction des données d'une instance du problème. Grâce à ce calcul, on dispose d'une garantie sur la qualité de toute solution obtenue : elle est faiblement optimale au sens de Pareto.

Cette fonction est étendue à la norme de Tchebychev pondérée augmentée, qui a la propriété d'être fortement croissante. Les solutions obtenues sont donc assurément Pareto-optimales.

La détermination des pondérations des deux critères, point important à fixer, est décrite. L'utilisation du point anti-idéal permet de ramener les deux critères à la même échelle afin de faciliter le paramétrage de la méthode proposée. Des tests effectués sur des instances du problème avec cette méthode mèneront à une évaluation de sa qualité.

5.3 Extension de l'algorithme mémétique

Dans cette section, on utilise les avantages de la métaheuristique présentée au chapitre 4 pour minimiser la norme de Tchebychev pondérée augmentée sur les trois critères considérés. On rajoute alors ici le nombre de déplacements de masques. En effet, l'algorithme mémétique, dans sa structure générique et modulaire, permet d'ajouter des critères indépendamment de la méthode d'optimisation utilisée. D'où l'idée d'analyser les résultats obtenus par cet algorithme sur la version multicritère du problème. Cela permet plusieurs observations :

- La corrélation entre les critères étudiés : des solutions efficaces sur un critère donné le restent-elles sur les autres ?
- L'efficacité de la méthode, vérifiable sur des instances de volume limité, pour lesquelles des solutions Pareto-optimales sont connues.
- Le paramétrage de la méthode doit-il s'adapter à la fonction objectif étudiée ?

La limitation de cette méthode est bien entendu son caractère approché. Aucune garantie d'optimalité n'existe pour les solutions obtenues, sans une méthode exacte fonctionnelle. Le fonctionnement de l'algorithme est le même qu'au chapitre 4, à l'exception du paramètre f_{obj}, qui intervient dans l'algorithme ÉVALUATION de la section 4.1.4. Cet algorithme permet de calculer la valeur d'un individu selon un critère spécifié en paramètre. Ici, il s'agit de la norme de Tchebychev pondérée augmentée dans sa version tricritère. On l'exprime ainsi :

$$s_{\lambda,r,\varepsilon} : \mathbb{R}^3 \to \mathbb{R}$$

$$(x_1, x_2, x_3) \mapsto s_{\lambda,r,\varepsilon}(x_1, x_2, x_3) = \max_{i=1,2,3} \{\lambda_i |x_i - r_i|\} + \varepsilon \sum_{i=1}^{3} \lambda_i |x_i - r_i|,$$

où x_1, x_2 et x_3 représentent ici les valeurs d'une solution selon les trois critères (respectivement le nombre de plaquettes produites dans un horizon de temps fixé, la somme pondérée des dates de fin et le nombre de déplacements de masques). Dans la section 5.4.2, des tests sont effectués sur les instances résolues en mono-critère à la section 4.2. On y observe l'intérêt de produire de bonnes solutions de compromis.

5.4 Résultats numériques

Cette section s'organise en deux parties. Premièrement, à la section 5.4.1, des fronts de Pareto sont générés sur des instances de tailles limitées à 10 tâches, pouvant être résolues par le programme linéaire mixte (FBCT-RF). On observe pour chaque instance, selon les valeurs données aux paramètres λ_1 et λ_2, les points obtenus dans l'espace des critères. L'impact du paramètre ε est également étudié. Puis, à la section 5.4.2, on présente des résultats sur les instances résolues par l'algorithme mémétique à la section 4.2 du chapitre 4, cette fois sur une version tricritère du problème, tenant donc compte du nombre de déplacements de masques. Un ensemble de points est généré pour chaque instance, dépendant des poids affectés aux critères $(\lambda_1, \lambda_2, \lambda_3)$.

5.4.1 Génération exacte du front de Pareto

Dans ce qui suit, des tests opérés au moyen de la méthode exacte à base de programmation linéaire en nombres entiers sur des instances de tailles modestes sont présentés. L'analyse des points du front de Pareto, obtenus en fonction des paramètres de la fonction d'agrégation, permet une bonne compréhension de cette technique. La section 5.2.3 justifie les choix faits pour les paramètres donnés à la fonction d'agrégation $s_{\lambda,r,\varepsilon}$. Les coordonnées du point de référence sont les valeurs optimales de chaque critère. Les instances traitées étant de taille limitée, elles peuvent être résolues par les formulations (FIT-1) et (FIT-2) présentées au chapitre 3. Les coefficients λ_1 et λ_2 sont fixés ainsi :

$$\lambda_1 = 1, \qquad \lambda_2 = c \times \frac{r_1 - x_1^N}{x_2^N - r_2}$$

où x_1^N est la valeur selon le premier critère de la solution optimale pour le deuxième critère et où x_2^N est la valeur selon le deuxième critère de la solution optimale pour le premier critère. On explique à la section 5.2.3 pourquoi ces coefficients sont pertinents. On fait varier la valeur de c entre 0 et 1 par pas de 0.01 puis entre 1 et 100 par pas de 1. Enfin, ε est fixé à 10^{-3}.

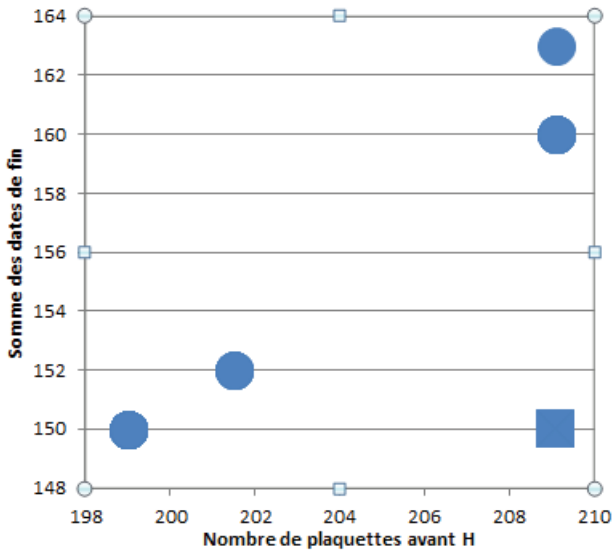

FIGURE 5.2 – Instance 1, front de Pareto généré par méthode exacte

Quelques instances du problème d'ordonnancement bicritère étudié sont traitées. Elles sont de petite taille car résolues au moyen d'une méthode exacte à complexité exponentielle, à savoir par *Branch and Cut* sur le programme linéaire en nombres entiers (FBCT-RF1). Le solveur utilisé est IBM ILOG CPLEX, version 12.5. Dans ce qui suit, on affiche toutes les solutions optimales au sens de Pareto générées avec les paramètres précédents. L'horizon de temps est fixé à $H = 25$ pour toutes les instances.

La figure 5.2 représente l'espace des critères pour l'instance 1. Les valeurs selon le premier critère (nombre de plaquettes produites dans un horizon de temps fixé) se trouvent en abscisses et les valeurs selon le second critère (somme des dates de fin d'exécution des tâches) se trouvent en ordonnées. Les points circulaires affichés représentent les solutions trouvées par la méthode exacte et le point rectangulaire représente le point idéal (regroupant les valeurs optimales selon les deux critères). On observe que le front de Pareto peut ne contenir que très peu de points, comme pour l'instance 1. En effet, sur 200 exécutions de l'algorithme de résolution, seuls trois points différents sont retournés.

Remarque 11. Notons que les points aux valeurs extrêmes (ayant la même abscisse que le point idéal) sont obtenus pour des valeurs nulles de λ_2, cas pour lequel la propriété de *forte croissance* se perd. Cela explique que ces points sont dominés dans le front de Pareto représenté. On les conserve quand même en gardant à l'esprit qu'ils sont obtenus pour un paramétrage particulier ($\lambda_2 = 0$). Les autres

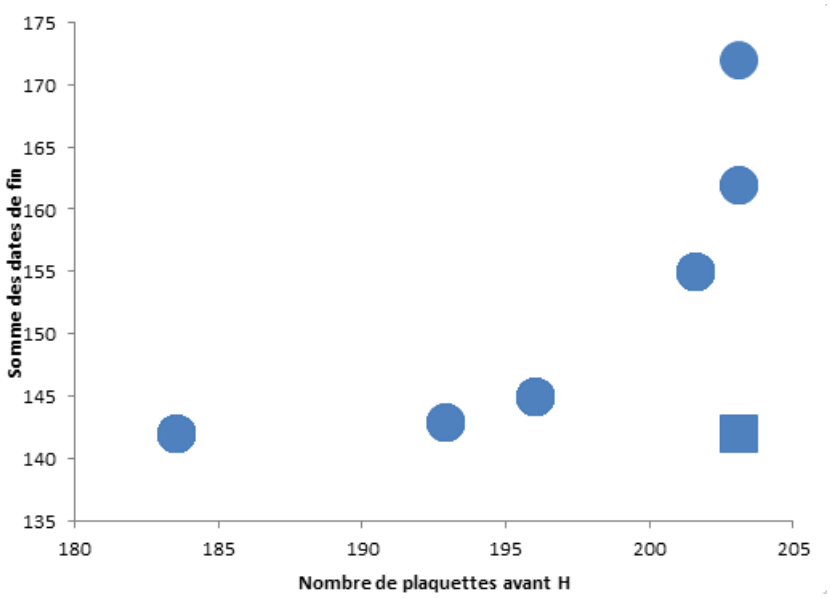

FIGURE 5.3 – Instance 2, front de Pareto généré par méthode exacte

points sont garantis Pareto-optimaux (forts).

La figure 5.3 représente le front de Pareto généré par la méthode exacte sur une deuxième instance, avec le paramétrage décrit en début de section. On observe que, hormis le point extrême évoqué à la remarque 11, toutes les solutions générées sont Pareto-optimales, et non *faiblement* Pareto-optimales.

Enfin, les figures 5.4 et 5.5 présentent les fronts de Pareto générés pour deux autres instances. Ces résultats confirment les propriétés de la méthode employée.

On peut comparer les résultats obtenus, par exemple sur la figure 5.2, avec l'ensemble obtenu par la minimisation de la norme de Tchebychev pondérée, c'est-à-dire pour $\varepsilon = 0$. La propriété de forte croissance est alors perdue. Ainsi, la figure 5.6 représente le front de Pareto avec deux solutions *faiblement* Pareto-optimales, désignées par les points blancs. Les autres sont garanties Pareto-optimales puisque présentes à la figure 5.2, trouvées pour $\varepsilon > 0$.

5.4.2 Solutions de compromis par méthode approchée

Les instances traitées dans la section 5.4.1 précédente sont de taille suffisamment limitée pour permettre d'obtenir des solutions optimales pour la minimisation de la distance au point idéal au sens de la norme de Tchebychev pondérée augmentée. L'utilisation de la métaheuristique introduite au chapitre 4 pour la minimisation de cette fonction objectif bicritère a prouvé son efficacité sur ces pe-

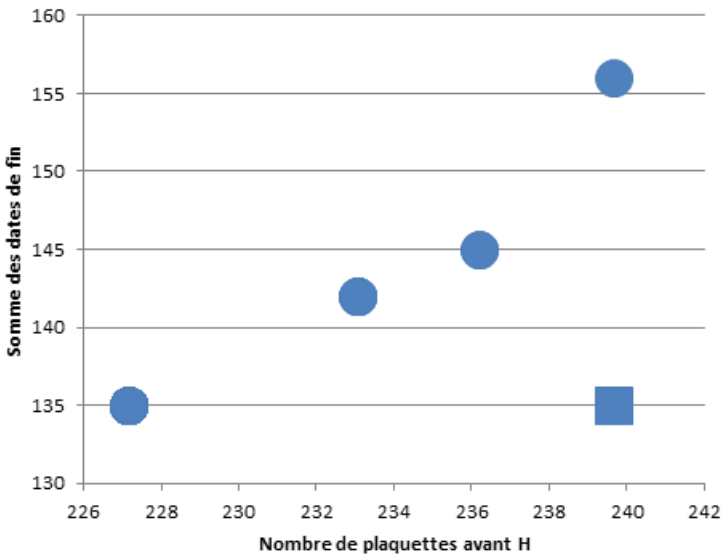

FIGURE 5.4 – Instance 3, front de Pareto généré par méthode exacte

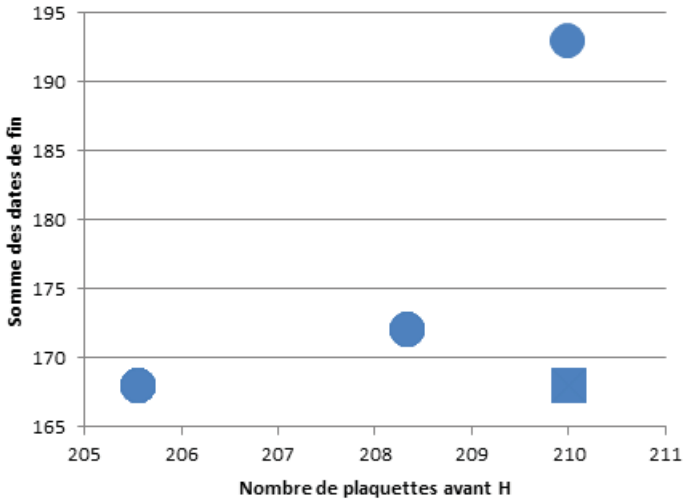

FIGURE 5.5 – Instance 4, front de Pareto généré par méthode exacte

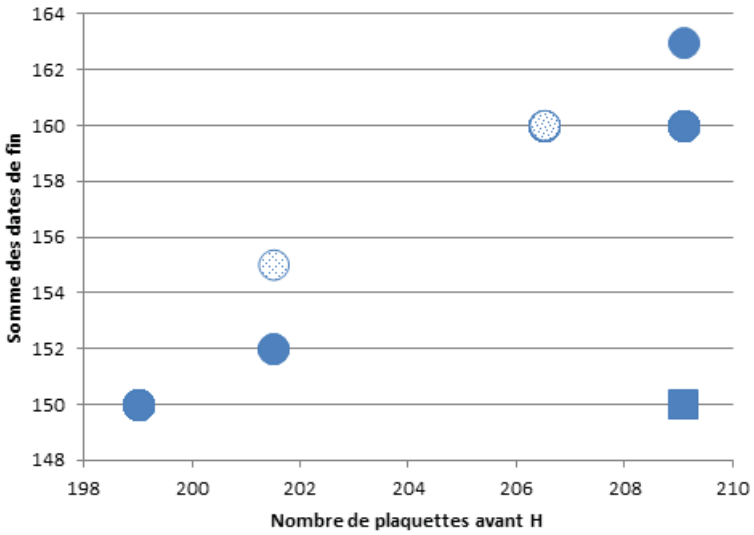

FIGURE 5.6 – Instance 1, front de Pareto généré par méthode exacte pour $\varepsilon = 0$

tites instances, puisque pour les mêmes paramètres λ, r et ε, elle fournit les mêmes points du front de Pareto que la méthode exacte utilisée en 5.4.1.

On aborde à présent la version *tricritère* du problème. La première interrogation concerne le paramétrage de la fonction d'agrégation utilisée ici et introduite à la section 5.3, à savoir

$$s_{\lambda,r,\varepsilon} : \mathbb{R}^3 \to \mathbb{R}$$

$$(x_1, x_2, x_3) \mapsto s_{\lambda,r,\varepsilon}(x_1, x_2, x_3) = \max_{i=1,2,3} \{\lambda_i |x_i - r_i|\} + \varepsilon \sum_{i=1}^{3} \lambda_i |x_i - r_i|,$$

où x_1, x_2 et x_3 représentent ici les valeurs d'une solution selon les trois critères. La section 5.2.3 explique comment fixer les paramètres dans le cas bicritère. Avec deux fonctions objectif, la détermination du point anti-idéal est facile à mettre en œuvre. Lorsque le nombre q de critères est supérieur ou égal à 3, comme on le verra, le calcul du point Nadir se fait de manière approchée.

Pour $s_{\lambda,r,\varepsilon}$, trois paramètres sont à déterminer.

- Le point de référence (r_1, r_2, r_3) dont les composantes sont les meilleures valeurs connues. Ici, la valeur r_3 peut s'obtenir pour toute taille d'instance par méthode exacte (voir résultats numériques à la section 3.4). Les valeurs de r_1 et r_2 s'obtiennent, selon la taille de l'instance, par méthode exacte ou approchée.
- Le coefficient d'augmentation ε qui doit être fixé à une valeur positive proche

de zéro, par exemple 10^{-k}, k entier suffisamment grand. En effet, toutes les remarques de la section 5.2.3 sur ce paramètre restent vraies pour $q \geq 3$.

– Les coefficients $(\lambda_1, \lambda_2, \lambda_3)$, à l'instar du cas bicritère, sont fixés à :

$$\lambda_1 = 1, \qquad \lambda_2 = c_1 \times \frac{r_1 - x_1^N}{x_2^N - r_2}, \qquad \lambda_3 = c_2 \times \frac{r_1 - x_1^N}{x_3^N - r_3},$$

où x_1^N, x_2^N et x_3^N sont les coordonnées du point anti-idéal, et c_1, c_2 sont les coefficients réels qui représentent la prépondérance respective des critères 2 et 3 sur le premier.

Le paramétrage des coefficients du vecteur de poids λ nécessite de générer le point anti-idéal. On a montré que ce point s'obtenait facilement à partir des solutions monocritère, dans le cas $q = 2$. Mais pour $q = 3$, les choses se compliquent.

Soient

– (x_1^1, x_2^1, x_3^1) la solution optimale selon la première fonction objectif, dans l'espace des critères,
– (x_1^2, x_2^2, x_3^2) la solution optimale selon la deuxième fonction objectif, dans l'espace des critères,
– (x_1^3, x_2^3, x_3^3) la solution optimale selon la troisième fonction objectif, dans l'espace des critères.

Pour fixer les idées, on remarque que $(r_1, r_2, r_3) = (x_1^1, x_2^2, x_3^3)$. On pourrait être tenté de poser

– $x_1^N = \min(x_1^2, x_1^3)$,
– $x_2^N = \max(x_2^1, x_2^3)$,
– $x_3^N = \max(x_3^1, x_3^2)$.

En effet, ce choix se justifie par les observations suivantes :

– *Sur les deux premiers critères*, tout point de l'ensemble des solutions efficaces domine le point (x_1^N, x_2^N),
– *Sur le premier et le dernier critère*, tout point de l'ensemble des solutions efficaces domine le point (x_1^N, x_3^N),
– *Sur les deux derniers critères*, tout point de l'ensemble des solutions efficaces domine le point (x_2^N, x_3^N).

Ces assertions valident les propriétés de point anti-idéal, en se restreignant à ces trois hyperplans de l'espace des critères. Mais la propriété n'est pas conservée sur l'espace tout entier. On se contente donc d'utiliser le point ainsi défini comme une *approximation* du point Nadir.

Dans les expérimentations qui suivent, réalisées sur des instances à 10 puis 50 tâches, les paramètres sont fixés comme on vient de le décrire. On fixe la durée de résolution à une minute pour chaque exécution de l'algorithme mémétique. Les coefficients c_1 et c_2 varient de $0,1$ à 5, par pas de $0,2$ dans l'intervalle $[0,1;1]$ et par pas de 1 dans l'intervalle [1;5]. Le paramétrage retenu est celui utilisé pour

$x_3 =$	6	5	4	3	2
	(**204**; **159**; **6**)	(**209**, **089**; **160**; **5**)	(**199**, **714**; **156**; **4**)	(**194**, **214**; **192**; **3**)	(**160**, **429**; **238**; **2**)
		(**201**, **5**; **152**; **5**)	(191, 5; 173; 4)	(**191**, **089**; **185**; **3**)	(**156**, **857**; **225**; **2**)
		(209, 089; 164; 5)	(189; 163; 4)	(183, 5; 215; 3)	(160, 429; 273; 2)
			(183, 286; 176; 4)	(172, 8; 213; 3)	(160, 429; 266; 2)
			(190, 786; 170; 4)	(178, 143; 199; 3)	(160, 429; 247; 2)
				(171; 230; 3)	(160, 429; 250; 2)
				(194, 214; 195; 3)	(160, 429; 258; 2)
				(188, 411; 211; 3)	(160, 429; 248; 2)
				(183, 5; 204; 3)	(160, 429; 255; 2)
				(171, 086; 220; 3)	(160, 429; 239; 2)
				(180, 375; 207; 3)	(160, 429; 260; 2)
				(188, 411; 207; 3)	(160.429; 246; 2)
				(188, 411; 197; 3)	(160, 429; 267; 2)
				(171, 5; 222; 3)	(160, 429; 242; 2)

TABLE 5.1 – Ensemble (partiel) de points de l'espace des critères, généré par l'extension de l'algorithme mémétique (10 tâches)

obtenir les meilleurs rendements de l'algorithme mémétique, notamments pour celles présentées à la section 4.2. Le but est d'observer les solutions obtenues par la méthode approchée proposée.

La table 5.1 présente les résultats obtenus pour 81 exécutions de l'algorithme (avec différentes valeurs de (c_1, c_2)). Tous les points générés ne sont pas affichés dans le tableau. On a séparé les points en colonne selon la valeur du troisième critère : le nombre de déplacements de ressources auxiliaires. Sur les 81 exécutions, on s'aperçoit que 45 points différents sont générés. Cela s'explique par la nature de l'instance qui, par sa taille, ne possède pas une variété importante de solutions efficaces. En comparaison, on ne génère jamais deux fois la même solution pour l'instance à 50 tâches. Si l'on restreint l'ensemble de solutions à considérer aux 45 solutions retournées, on remarque que 37 sont faiblement Pareto-optimales et 8 sont Pareto-optimales. Ces dernières sont représentées en gras dans le tableau. Cela implique deux choses :

- D'une part, aucune solution n'est strictement dominée dans cet ensemble. Cela valide la qualité de la fonction d'agrégation et de la méthode approchée sur des petites instances.
- D'autre part, le nombre de solutions faiblement efficaces étant assez important, l'algorithme mémétique n'est bien qu'une méthode approchée. Il peut fournir des solutions faiblement dominées sur des instances de petite taille.

La table 5.4.2 présente les résultats des expérimentations lancées sur une instance du problème à 50 tâches. 101 points ont été générés par cette méthode, dont seuls 16 apparaissent Pareto optimaux (au sens strict) dans cet ensemble. Les autres solutions sont en général faiblement Pareto-optimales. Ici, on observe

$x_3 \leq$	20	25	35
	(**839**, **333**; **3415**; **13**)	(**881**, **667**; **2711**; **21**)	(**989**, **278**; **2538**; **26**)
	(**832**, **681**; **3155**; **14**)	(**906**, **5**; **2883**; **21**)	(**987**; **2242**; **28**)
	(**912**, **75**; **2897**; **16**)	(**888**, **611**; **2730**; **22**)	(**1011**, **53**; **2380**; **29**)
	(**881**, **819**; **2797**; **19**)	(**925**, **292**; **2753**; **23**)	(**999**; **2352**; **33**)
	(**925**, **048**; **2965**; **20**)	(**915**, **5**; **2690**; **24**)	(943; 2931; 26)
	(627; 4081; 13)	(**940**, **056**; **2903**; **23**)	(976, 333; 2769; 26)
	(675, 661; 4052; 14)	((**970**, **651**; **2636**; **25**))	(962, 375; 2661; 26)
	(591, 5; 4289; 14)	(913, 833; 2895; 24)	(971, 944; 2638; 28)

TABLE 5.2 – Ensemble (partiel) de points de l'espace des critères, généré par l'extension de l'algorithme mémétique (50 tâches)

que la méthode utilisée, avantageuse pour calculer des solutions efficaces, nécessite un bon paramétrage et, encore mieux, un point de comparaison.

Il serait en effet très intéressant d'évaluer la qualité des ensembles de points obtenus, en comparant avec d'autres méthodes d'optimisation multicritère pour ces instances. Cela permettrait d'établir, au moyen d'indicateurs tels que la *Net Front Contribution* (NFC), le réel apport de cette méthode, difficile à évaluer car sans garantie de performance en théorie. Ce point constitue une perspective de la suite de ce travail.

5.5 Conclusion

Dans ce chapitre, on étudie le problème d'ordonnancement à machines parallèles différentes éligibles, avec ressources auxiliaires et temps de setup dépendant de la séquence et de la machine sous l'aspect multicritère. Un état de l'art sur les concepts et les modèles utilisés dans la littérature est présenté. Une méthode exacte est proposée pour la résolution du problème avec deux critères, le nombre de plaquettes produites dans un horizon de temps fixé et la somme des dates de fin d'exécution. Le modèle utilisé est basé sur une fonction d'agrégation, linéarisée au sein du PLNE présenté au chapitre 3. Cette fonction, la norme de Tchebychev pondérée augmentée, a de nombreux avantages et permet notamment d'atteindre l'ensemble des solutions efficaces du problèmes. Les inconvénients sont liés à la détermination des paramètres de la norme de Tchebychev pondérée augmentée. En effet, il s'agit d'une métrique à un point de référence, dont les coordonnées peuvent être déterminées, de préférence par résolution monocritère du problème. Par ailleurs, la méthode exacte est à complexité exponentielle (voir chapitre 3) et seules des instances de taille réduite peuvent être résolues.

L'algorithme mémétique présenté au chapitre 4 peut être étendu à la norme de

Tchebychev pondérée augmentée, cette fois dans sa version *tricritère*. L'avantage de cette méthode est son temps d'exécution, qui permet de traiter des instances de taille industrielle. Cependant, aucun garantie n'est fournie sur le caractère Pareto-optimal des solutions retournées. La poursuite de ces travaux nécessite d'avoir d'autres techniques de génération des points du front de Pareto, et les comparer au moyen d'indicateurs couramment utilisés en optimisation multicritère (voir par exemple [85]).

Par ailleurs, aucune méthode exacte n'a été mise en œuvre pour étudier les trois critères conjointement. Or, la corrélation entre ces trois objectifs a un réel intérêt. La méthode exacte pour optimiser le nombre de déplacements de masques engendre des solutions qui négligent fortement les deux autres critères qui eux, sont réguliers. Un point important à noter ici est que divers degrés de liberté sont offerts par ces solutions. En effet, les tâches partageant la même ressource auxiliaire (ou masque) et affectées à la même machine peuvent être exécutées dans un ordre quelconque et ne modifient pas la valeur de la solution selon le troisième critère (nombre de déplacements de masques). C'est pourquoi la recherche, dans le sous-espace de solutions correspondant, de solutions Pareto-optimales au sens des deux autres critères (par une méthode adaptée aux critères réguliers) produirait une solution Pareto-optimale du problème tricritère. Ce type de méthode à deux phases pourrait alors tirer parti des résultats connus sur le problème, formulés aux chapitres 3 et 4.

Chapitre 6

Mise en œuvre industrielle

Les travaux présentés dans ce manuscrit, notamment ceux du chapitre 4, ont reposé sur un projet industriel visant à automatiser les décisions d'ordonnancement dans l'atelier de photolithographie du site d'un fabricant de semi-conducteurs (site de Rousset de *STMicroelectronics*). Il s'agit d'un des objectifs du projet européen *INTEGRATE*, visant entre autres à améliorer la connaissance et l'efficacité des procédés de fabrication dans le domaine de la fabrication des semi-conducteurs. L'objectif est de proposer un logiciel d'ordonnancement des lots entrant dans l'atelier, capable de déterminer périodiquement une organisation optimisée de la production tenant compte de données en temps réel. Plusieurs problématiques se posent dans ce contexte.

- D'une part, la phase de validation de l'outil, à travers un protocole expérimental. Cette méthode est issue d'une discussion avec les industriels sur la mise en place d'indicateurs fiables et de procédures pertinentes de tests.
- D'autre part, l'industrialisation de l'outil et les aspects d'intégration dans un système existant. Parmi ces aspects, on compte la communication avec les modules d'extraction dans la base de données, le pré-traitement, l'affichage dans l'atelier, la gestion des imprévus ou encore la configuration du logiciel.

On n'omettra pas d'évoquer les conséquences de cette intégration dans la gestion de l'atelier. Par exemple, les opérateurs seront moins sollicités pour certaines tâches sans valeur ajoutée comme le déplacement de masques. La réorganisation de la zone de photolithographie s'inscrit dans le cadre plus large de l'automatisation de l'usine. C'est aussi l'optique visée dans ce projet. Dans ce qui suit, les objectifs, le cadre et la problématique, sont décrits à la section 6.1. Les aspects les plus complexes du travail, comme la gestion des données d'entrée et la validation des résultats des tests seront respectivement évoqués aux sections 6.2 et 6.3. La description de la mise en œuvre industrielle en section 6.4 viendra clore ce chapitre.

6.1 Conduite du projet

6.1.1 Contexte industriel

Le projet de création d'un logiciel d'ordonnancement optimisé des lots dans l'atelier de photolithographie s'inscrit dans le cadre du projet européen *INTEGRATE* et a démarré en 2013. L'objectif est d'intégrer une méthode automatique de planification optimisée à court terme des lots dans l'atelier de photolithographie au sein de l'unité de fabrication de *STMicroelectronics* basée sur le site de Rousset, dans le Sud de la France. Une première version d'un logiciel, développé par l'École Nationale Supérieure des Mines de Saint-Étienne en collaboration avec le site *STMicroelectronics* de Crolles, en France, avait été testée. Toutefois, cet outil n'a pas été intégré. L'abandon de cette solution a été un choix stratégique de la compagnie, qui a préféré opter pour un logiciel commercial pour son site de Crolles. Cette décision a été motivée par la volonté de disposer d'une garantie de support et de maintenance du logiciel. Le projet de l'adapter au site de Rousset en vue de son utilisation est né d'un réel besoin d'accroître les performances de production, dans une unité où le débit d'entrée des lots est très soutenu.

La nécessité de se concentrer en priorité sur la photolithographie n'est pas due au hasard. Il s'agit d'un atelier critique dans l'usine, du fait du coût des machines dédiées à cette étape ou encore de la gestion des ressources auxiliaires que constituent les masques. Ainsi, une amélioration des temps de cycle à cette étape est globalement bénéfique.

Cependant, plusieurs difficultés rencontrées dans l'adaptation ont conduit à repenser la structure générale du logiciel et à réécrire un programme en y ajoutant l'algorithme d'optimisation présenté au chapitre 4. Dans la suite, on décrira les adaptations nécessaires à la prise en compte de caractéristiques supplémentaires. Ces adaptations, liées aux contraintes de la zone, n'avaient pas été retenues dans la modélisation et l'étude du problème d'ordonnancement défini au chapitre 1. Elles sont importantes dès lors que l'optimisation s'applique au contexte réel.

6.1.2 Adaptation du programme

Les données du problème défini au chapitre 1 sont communes à tous les ateliers de photolithographie en fabrication de semi-conducteur, même s'ils ne couvrent pas tous les détails. Il a fallu rajouter des contraintes additionnelles, listées ci-dessous.

Bibliothèque de masques Les machines ont une capacité limitée de stockage de masques. En effet, si un masque est requis pour l'exécution d'une tâche, il faut vérifier s'il lui reste de la place.

Boîtes de masques Les masques sont stockés dans des conteneurs communs et chaque conteneur peut en renfermer 5 au plus. Cela implique qu'il peut exister des contraintes d'utilisation non simultanée des masques. En d'autres termes, utiliser un masque en bloque 4 autres, ceux qui se trouvent dans le même conteneur. Cette contrainte n'existe notamment pas sur d'autres sites de *STMicrolectronic* comme celui de Crolles, où chaque masque est déplacé individuellement. On verra plus tard que des simulations ont été effectuées dans ce sens pour évaluer la pertinence de la suppression de ces conteneurs partagés.

Disponibilité des lots Les lots ne sont pas tous disponibles à la même date.

Enchaînement des lots Les machines disposent de ports de chargement multiples (jusqu'à 4 en général) qui permettent de gérer simultanément plusieurs lots.

Temps de transport Ils concernent toutes les ressources mobiles de l'atelier, à savoir les lots et les masques, dans leurs conteneurs. Ils sont jugés négligeables dans la modélisation précédente, hormis pour les masques, où une constante est fixée pour les temps de déplacement. Dans la réalité, il faut considérer une matrice de temps de transport. Déterminer ces temps a priori n'est pas aisé, et des approximations basées sur des statistiques sont fournies par les industriels.

Maintenances Les machines ont des périodes de maintenance. Dans ces intervalles, aucune tâche ne peut se terminer. Ces périodes peuvent aussi correspondre à des imprévus, en particulier des pannes.

Enchaînement des lots Les temps d'exécution des lots sur les machines sont difficiles à prévoir, donc à fixer a priori pour le fonctionnement d'un logiciel d'ordonnancement. Le modèle le plus synthétique retenu pour intégrer cette donnée est de considérer deux valeurs :

- Le *C-Time*, qui est le temps total passé par une plaquette entre son entrée et sa sortie de la machine ; c'est en d'autres termes le temps de séjour d'une plaquette dans la machine.
- Le *P-Time*, temps passé entre l'entrée dans la machine de deux plaquettes consécutives du même lot ; c'est l'interarrivée entre deux plaquettes.

Le temps d'exécution d'un lot sur une machine, dont le *C-Time* est c, le *P-Time* est p et le nombre de plaquettes est w, est de :

$$p \times (w - 1) + c.$$

La simultanéité d'exécution de plusieurs lots sur la même machine est autorisée par la présence de ports de chargement, comme expliqué ci-dessus. Mais il y a quand même des règles à suivre. Un lot ne peut débuter sur la

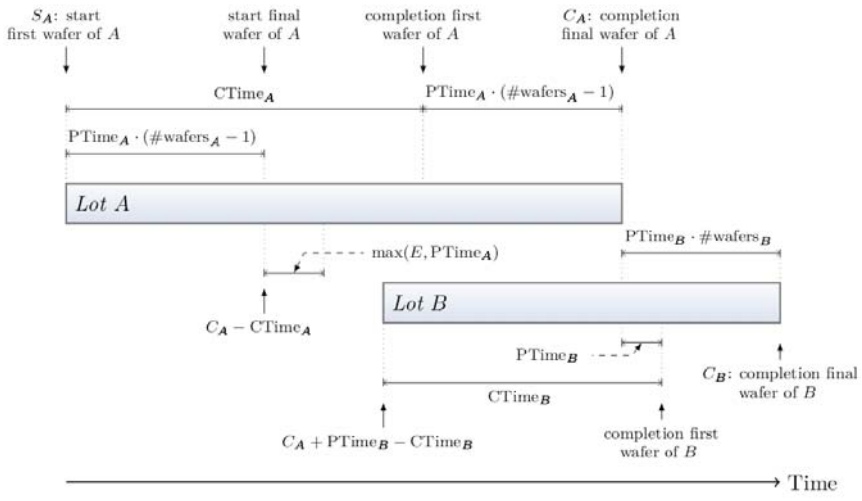

FIGURE 6.1 – Passage des lots dans une machine de photolithographie (Schéma simplifié)

machine que si les plaquettes du lot qui le précède sont toutes dans la machine. Avec les notations précédentes, ce temps correspond à $p \times (w-1)$. Le temps de préparation (*setup*) s'ajoute à ce délai. De même, les plaquettes du lot ne peuvent commencer à sortir de la machine que si toutes celles du lot précédent sont sorties. Le schéma de la figure 6.1 présente la succession de lots dans un équipement de photolithographie. Ce fonctionnement très particulier de l'enchaînement des lots a été pris en compte dans le logiciel afin de permettre un calcul le plus fidèle possible des dates de début des lots.

Le degré de détail retenu dans le modèle découle d'une discussion active auprès des ingénieurs et autres spécialistes de l'atelier de photolithographie. On peut s'apercevoir de la divergence de points de vue, au sein d'un même service, dans la façon de percevoir l'importance de certaines contraintes. Ces divergences s'expliquent aussi par les différences de perception d'un logiciel d'optimisation. Est-ce un outil d'aide à la décision, censé donner une direction et appuyer une stratégie globale, ou un moyen de simuler la réalité avec précision, afin de prédire les indicateurs de la production ? Cette question est souvent revenue dans le processus de construction du modèle.

6.1.3 Étapes du projet

Pour atteindre l'objectif final, plusieurs étapes importantes ont été nécessaires. Il faut que le programme tienne bien compte des contraintes de l'atelier, notam-

ment celles qui sont spécifiques au site de Rousset de *STMicroelectronics*. Ces contraintes sont fixées à l'issue d'une discussion avec les industriels. Ces derniers conviennent des notions les plus importantes à considérer. Mais avoir un programme tenant compte des réalités ne suffit pas. Il faut optimiser et qui dit optimisation dit *indicateurs de performance*. Ce point est essentiel, notamment pour la validation de la qualité de l'algorithme.

L'étape de validation, qui a été la plus longue dans notre cas, repose sur un dialogue cohérent sur le modèle (contraintes et critères retenus) avec les industriels. Par ailleurs, le pré-traitement des données d'entrée dans un format fixé, tâche qui incombe à l'entreprise, est important puisqu'il sera utile dans la phase d'industrialisation. Dans la suite de ce chapitre, on évoque cette étape et les décisions prises pour valider concrètement les performances du logiciel d'ordonnancement optimisé, sur des données réelles.

La dernière phase, celle de la mise en œuvre, pose des questions de structure et d'intégration dans le système d'information en temps réel de la production. Ces questions sont essentielles pour la robustesse de la solution logicielle. Une bonne intégration assure un fonctionnement pérenne. Il faut considérer la réticence des industriels à bouleverser leur mode de fonctionnement. Cet aspect sera présenté de façon plus détaillée à la fin du chapitre. Il y a un autre aspect concerné par ce bouleversement : la gestion des opérateurs dans un système où la part d'automatisation aura sensiblement augmenté. La manière de gérer ce changement est d'un grand intérêt dans notre cas. Il s'agit d'analyser comment s'inscrivent les solutions issues de la recherche opérationnelle dans des systèmes industriels et quels sont les impacts à tous les niveaux.

6.2 Accès aux données

Préalablement à l'étape de validation proprement dite, une longue période a été consacrée à des échanges sur le format d'entrée des données. De façon générale, selon le volume et la diversité des informations requises, l'automatisation et la standardisation du procédé peut prendre beaucoup de temps en entreprise. Plusieurs services sont parfois concernés, il n'est pas évident de savoir où chercher l'information, etc. Dans notre cas, cette étape préalable à la validation a pris quelques mois. Si ce travail revenait à l'entreprise, il était nécessaire de vérifier l'intégrité et la cohérence des données fournies. C'est pourquoi un module de lecture de données a été élaboré, avec des rapports d'erreurs sur chaque fichier d'entrée, couvrant un large ensemble d'exceptions possibles. Cette vérification automatique a donc permis de valider la cohérence et le traitement des instances industrielles.

Il faut noter que les difficultés dans l'extraction des données venaient souvent d'informations incomplètes sur certains lots, équipements ou masques. Une autre

source importante d'erreurs provenait de la mise en commun des informations arrivant de bases de données différentes. Par ailleurs, certains pré-traitements mis en place entre les phases d'interrogation des bases de données et de mise au format d'entrée du logiciel, ont dû se mettre en place. L'organisation de ces informations est néanmoins une étape importante du travail puisqu'elle permet d'entrevoir la manière d'organiser les données dans le cadre d'une application plus large, à savoir des outils d'optimisation de l'ordonnancement dans divers ateliers, une gestion plus globale d'une zone de fabrication ou encore la communication des flux de données entre ateliers.

On peut citer un exemple concernant le fichier qui contient toutes les données relatives aux lots. Ce fichier contient une ligne pour chaque lot, et à chacune de ces lignes, le programme doit pouvoir lire les données vitales à son bon fonctionnement, sous peine d'ignorer la ligne en question et de retourner un message dans un fichier de rapport d'erreurs. Ces champs vitaux sont notamment l'*étiquette* du lot (nom ou *label*), le nombre de plaquettes du lot, sa classe de priorité, son masque requis, ou encore sa date de disponibilité. Cependant, d'autres champs sont présents, nécessaires dans d'autres ateliers ou pour d'autres applications. On rappelle qu'un lot passe dans différentes zones au cours du processus de fabrication et que les étapes sont nombreuses. Une communication entre ces zones est importante et c'est dans le souci de standardiser les échanges et la production de données que le format associé au lot comprend des champs supplémentaires non requis dans l'outil d'ordonnancement optimisé en photolithographie.

6.3 Simulation et validation des résultats

6.3.1 Comparaison en contexte similaire

En dépit du niveau de détail important apporté au modèle des données d'entrée, la comparaison entre un ordonnancement réalisé dans un contexte réel et un autre simulé à l'aide d'un logiciel d'optimisation, s'avère difficile à effectuer. Par exemple, il peut y avoir un fonctionnement différent de la machine, les temps de chargement et de déchargement des lots peuvent diminuer ou augmenter et les durées d'exécution peuvent par conséquent changer en réalité. Les dates de début et de fin d'exécution des lots sont calculés dans l'optimisation pour fournir une estimation de la qualité de la solution. Cette estimation sera d'autant plus fidèle qu'on alimentera le modèle de données détaillées. C'est pourquoi la comparaison s'effectue de la façon suivante :

- Un module de lecture de la *Tool Task List*, ou *TTL* est mis en place au sein du logiciel. La TTL est une liste ordonnée de lots à exécuter pour chaque machine. Un mode de lancement du logiciel est donc créé, dans lequel au-

cune décision d'ordonnancement n'est prise. Il s'agit ici, en mode TTL, de reproduire l'ordonnancement indiqué par cette liste, en calculant les dates de début et de fin de chaque lot à partir des contraintes du modèle.
- On lance le logiciel d'ordonnancement en mode TTL, sur un critère d'optimisation fixé.
- On lance le logiciel d'ordonnancement en mode normal d'optimisation sur le même critère.
- On compare les deux solutions obtenues selon ce critère.

L'objectif est de s'affranchir des aléas et imprévus du contexte réel pour comparer deux ordonnancements. Les simulations ont été faites sur des journées de production déjà réalisées, avec des TTL ainsi disponibles. D'importants volumes de données (entre 1200 et 1400 lots sur 24 heures) ont été traités par le logiciel d'ordonnancement.

6.3.2 Simulation de nouveaux scénarios

La phase de tests a dévoilé d'autres avantages de l'outil quant aux indications qu'il pouvait donner sur le fonctionnement de l'atelier. Le niveau élevé de détails apportés au modèle des données d'entrée a permis d'étudier certains cas particuliers, notamment le cas des conteneurs individuels. Pour déterminer si un changement dans ce sens apporterait un gain significatif d'efficacité, plusieurs tests ont été lancés avec le scénario excluant la contrainte de conteneurs de masques partagés. Ainsi, chaque masque est utilisé individuellement. Lorsqu'il est utilisé, il ne bloque pas d'autres masques qui partagent le même conteneur. Les résultats ont démontré des gains intéressants sur certains indicateurs.

Rappelons le choix des fonctions objectif retenues pour l'étude du problème.

1. **Le nombre de plaquettes traitées dans une fenêtre de temps fixée**. Ce critère concerne directement la productivité puisqu'on vise ainsi à maximiser le nombre de plaquettes traitées et le fait de fixer une fenêtre de temps permet de considérer, dans une utilisation temps réel, une actualisation des données et les lots qui arrivent dans l'atelier de façon dynamique. Ce critère apparaît comme le plus important et le plus adapté à l'utilisation finale du logiciel.
2. **La somme pondérée des dates de fin d'exécution des lots**. Dans notre cas, les pondérations sont de trois sortes, selon la priorité du lot. Un poids ω_1 est accordé à la priorité faible, un poids ω_2 à la priorité moyenne et un poids ω_3 à la priorité forte, avec $\omega_1 < \omega_2 < \omega_3$. Par ailleurs, une modification importante de cette fonction objectif par rapport aux chapitres précédents repose sur la définition du *temps de cycle*. On ne considère pas la date de fin d'un lot, mais le temps écoulé entre sa date de disponibilité (date

d'arrivée dans l'atelier de photolithographie) et sa date de fin d'exécution. Précédemment, on a considéré que les tâches étaient disponibles à la date 0 d'où l'équivalence des deux critères. En notant γ_i la classe de priorité du lot i et r_i sa date de disponibilité, le temps de cycle moyen, aussi appelé *X-Factor* dans notre cas, est donné par la formule :

$$XFactor = \frac{\sum_{i \in J} \omega_{\gamma_i} \frac{C_i - r_i}{\min_{j \in \mathcal{M}_i} \rho_{ij}}}{\sum_{i \in J} \omega_{\gamma_i}}.$$

Ici, le quotient $\frac{C_i - r_i}{\min_{j \in \mathcal{M}_i} \rho_{ij}}$ vaut 1 dans le meilleur des cas, c'est-à-dire celui où le lot passe immédiatement sur sa machine au moment où il est disponible et sa machine est la plus rapide des machines qualifiées pour ce lot. On veut donc minimiser ce ratio, d'autant plus quand le lot est prioritaire.

3. **Le nombre de déplacements de masques**, qui dans le contexte industriel devient le nombre de déplacements de conteneurs. On observe sans surprise que dans le scénario où chaque masque possède son propre conteneur, la valeur de cet indicateur augmente, puisque le nombre de conteneurs augmente. Ce scénario pose la question de la gestion de ces conteneurs. Une augmentation de leur nombre engendre inéluctablement la question du stockage et conséquemment d'un module de gestion des conteneurs de masques. Les contraintes limitant la capacité des équipements amènent à considérer des lieux de stockage dédiés ainsi qu'une capacité associée à gérer. Ce cas est à l'étude puisqu'envisager de modifier le fonctionnement d'un atelier en y apportant de l'automatisation s'accompagne d'autres modifications de type structurel et celle-ci en est une.

Ces trois indicateurs ont été retenus pour analyser les performances du logiciel d'ordonnancement optimisé et comparer avec l'efficacité du fonctionnement actuel. D'autres indicateurs, comme le *Process Equipment Throughput*, ou *PET*, concernant le débit moyen de sortie de plaquettes des machines, ont été analysés.

L'optimisation dont on parle ne concerne que l'atelier de photolithographie. Il ne faut pas oublier le reste de l'usine malgré sa complexité et la diversité de ses ateliers. Ainsi, afin de réguler les flux sortant de la photolithographie vers les autres étapes de production, des *Focus*, ou *Targets* sont utilisés. Ce sont des objectifs (ou consignes) à court terme fixés par tranche de 24 heures, sur des plaquettes d'un certain type. Un nombre de plaquettes à traiter par l'atelier est fixé en guise d'objectif pour les 24 prochaines heures et ces consignes sont là pour guider la production dans le contexte général de la zone de fabrication. Intégrer cet aspect relève donc de la prise en compte globale des flux incidents aux autres ateliers. Un indicateur de satisfaction de ces consignes est donc ajouté à l'optimisation. L'un des plus grands avantages de la méthode d'optimisation utilisée, à savoir l'algorithme

mémétique, est sa modularité et la possibilité d'y ajouter de nouvelles fonctions objectif sans avoir à modifer la structure de la méthode.

On note ici que les informations relatives aux consignes de type *Focus* ou *Target* sont à considérer comme des objectifs permettant d'harmoniser la production inter-atelier. On cherche alors à éviter d'être en sur-production par rapport aux ateliers en aval, et les outils d'optimisation et d'aide à la décision doivent tenir compte des flux attendus en sortie de la zone. Optimiser le temps de cycle dans un atelier est un excellent objectif à l'échelle locale, mais peut ne produire qu'un temps de cycle médiocre sur l'ensemble du procédé de fabrication.

6.3.3 Analyse des résultats

Il existe des différences majeures entre les étapes de validation et l'utilisation finale de l'outil d'optimisation.

- Tout d'abord, le volume des données traitées est radicalement différent. Les tests effectués dans un premier temps s'opèrent sur des données volumineuses (ensemble d'une journée entière réalisée) alors qu'à terme, l'optimisation se fera périodiquement sur des tranches temporelles bien plus réduites (de l'ordre d'une ou deux heures selon le débit d'arrivée des lots et l'en-cours existant, qu'on appelle aussi le *Work In Progress*, ou *WIP*).
- L'exigence sur la précision des données d'entrée du logiciel n'est pas la même. Il faudra être rigoureux dans l'utilisation finale, alors qu'on peut se permettre quelques approximations à l'étape de tests. Par exemple, les temps de préparation (ou setup) pourront être calculés de façon plus grossière, de même pour les temps de transport. En réalité, l'étape de validation s'est construite progressivement. Les données au départ ne contenaient ni le fichier de *Focus/Targets*, ni les temps de transport, ni les temps de setup.
- L'étape de validation ouvre le champ à des études plus poussées afin de simuler différents scénarios. Certains scénarios, pour lesquels le site de Rousset de STMicroelectronics a exprimé un grand intérêt, ont justement été étudiés, notamment le cas où les masques ne partagent pas de conteneurs. On peut analyser divers indicateurs afin de juger de la cohérence des résultats.

Le tableau 6.1 présente des résultats (normalisés pour des raisons de confidentialité) sur un ensemble de données réelles de la zone de photolithographie traitées par notre logiciel d'optimisation. Les lignes présentées permettent de rendre compte de la précision, la cohérence, l'efficacité et des possibilités du logiciel. En effet, la première ligne présente les valeurs des trois critères considérés, calculées à partir de la TTL, i.e. l'affectation et la séquence réelle des lots dans l'atelier. L'écart entre le nombre réel de plaquettes traitées et celui calculé par le logiciel est inférieur à 1,5%. Les trois lignes suivantes présentent cette fois les résultats fournis

Scénario	Nombre de plaquettes	X-Factor	Transferts de masques
Mode TTL (réel $X + 1,5\%$)	X	Y	Z
Solveur TriCritère	$X + 2,2\%$	$Y - 6,5\%$	$Z - 16,1\%$
Solveur Plaquettes	$\mathbf{X + 2,8\%}$	$Y + 78,8\%$	$Z + 41,1\%$
Solveur X-Factor	$X - 0,2\%$	$\mathbf{Y - 16,9\%}$	$Z + 34,3\%$

TABLE 6.1 – Résultats sur des données réelles, fournis par le logiciel d'optimisation

Scénario	Nombre de plaquettes	X-Factor	Transferts de masques
Mode TTL (réel $X + 1,5\%$)	X	Y	Z
SinglePod TTL	X	Y	$Z + 19,9\%$
SinglePod TriCritère	$X + 3,9\%$	$Y - 16,8\%$	$Z + 5,4\%$
SinglePod Plaquettes	$X + 3,6\%$	$Y + 72,4\%$	$Z + 53,4\%$

TABLE 6.2 – Résultats du scénario avec conteneurs individuels de masques, fournis par le logiciel d'optimisation

par les décisions du logiciel, en optimisant les trois critères simultanément, via une fonction d'agrégation simple, la somme pondérée des critères, puis en maximisant le nombre de plaquettes traitées sur la fenêtre fixée, et enfin en minimisant le critère du X-Factor, associé aux temps de cycle des lots pondérés par leur priorité. Les valeurs surpassent les chiffres fournis par la TTL, notamment pour le tricritère, soulignant ainsi l'efficacité de l'optimisation. La cohérence est visible en analysant les valeurs obtenues selon le critère fixé. La meilleure valeur de X-Factor est atteinte lorsque ce critère est optimisé, de même pour le nombre de plaquettes. Le choix a été fait de ne pas optimiser le nombre de transferts de masques seul, étant donné le peu de pertinence de la solution obtenue. Ce tableau ne présente les résultats que sur un ensemble limité de données. Un grand nombre de journées de production ont été analysées et celle-ci est l'une des plus représentatives. Peu d'imprévus y ont été enregistrés et la quantité de lots à traiter est suffisamment importante pour permettre une marge d'amélioration fournie par l'optimisation.

Le tableau 6.2 témoigne des possibilités apportées par le logiciel, qui permet de simuler de nouveaux scénarios de production. Ici, le cas de conteneurs de masques individuels est analysé. On constate sans surprise que le nombre de transferts de masques augmente puisque le nombre de conteneurs augmente par définition. Le point intéressant ici est l'amélioration observée sur les autres critères.

L'étape de validation, très longue, s'est étendue sur une dizaine de mois. À son issue, un accord pour mise en production a été obtenu, suite aux bons résultats fournis par les simulations. L'étape de mise en œuvre industrielle a consisté en la transformation du prototype en un logiciel exécutable en condition réelle et intégré aux systèmes de *STMicroelectronics*.

6.4 Mise en œuvre industrielle

La mise en production est une tâche bien délicate. De nombreux choix dans la conception du programme, jusqu'à la plate-forme de développement choisie, vont influencer la vitesse et la fluidité de cette phase. De nombreuses questions se posent à l'entame de ce processus :

– Le niveau de criticité de l'application, qui dans ce cas, est élevé. Cet indicateur donne une idée de la gravité d'un éventuel arrêt ou dysfonctionnement et par conséquent de l'urgence d'une intervention. Il faut être capable de réagir vite dans une situation de ce type, d'où l'importance de décider d'un protocole d'action dès le départ pour faire face à ce type d'évènement. Ici, comment gérer un arrêt de fonctionnement du moteur de planification des lots en photolithographie ? Comment détecter les sources du problème ? Quelle est la liste de vérifications à faire pour les détecter ? Puis selon le cas, quelle solution employer ?

- Il faut spécifier les acteurs de ces protocoles. La maintenance du logiciel pour toute évolution ou intervention éventuelle est une question cruciale. Elle doit être réglée avant toute chose. Le niveau de compétence et d'expertise requis pour ce type de tâches est élevé, la présence d'éléments bien formés sur l'outil est indispensable.
- Pour cette raison, le transfert de connaissance est organisé au cours de plusieurs réunions. Cette étape préalable a pour but de donner toutes les clés de compréhension des concepts utilisés dans le logiciel. Cela permet à des éléments internes de l'entreprise d'avoir des notions sur la façon de comprendre la méthode algorithmique utilisée, mais aussi le code source.
- Le code source doit être lisible, bien commenté, modulaire et d'une relative portabilité. L'efficacité en temps d'exécution est un facteur non négligeable dans ce cas d'application. Son fonctionnement permanent, adaptatif à la situation et aux changements des statuts des lots et des équipements dans l'atelier, rendent son utilisation fréquente. Le rythme des appels au moteur d'optimisation sera soutenu, il faut alors être particulièrement attentif à ces aspects. Par ailleurs, certaines conventions d'écriture et de conception en programmation orientée objet, sont propres au service informatique de l'entreprise. S'adapter à ces schémas induirait un gain substantiel de temps. Malheureusement, dans notre cas, le travail a commencé bien en amont et le service informatique est intervenu dans le processus à l'étape d'intégration, quand le code source était déjà sous forme assez aboutie. Cependant, ces fichiers respectaient déjà une grande partie des règles de programmation de base, et dans une certaine mesure, de bon sens.
- La migration et l'adaptation à d'autres unités de production ont été évoquées. Cela impliquerait un travail de mise en conformité, non seulement aux systèmes internes des autres usines, mais aussi à leurs contraintes de fonctionnement au niveau de l'atelier de photolithographie. La modification du code source qui en découle, bien que probablement modérée, serait un travail nécessaire et la détermination des contributeurs est importante si l'on veut perdre le moins de temps possible.

Plusieurs modules vont communiquer avec le logiciel d'optimisation :

- Le module d'extraction dans la base de données, qui permet d'accéder aux informations importantes en temps réel, sous format *brut*.
- Le module de pré-traitement, en langage R, pour organiser ces données au format spécifié par le manuel d'utilisateur du logiciel.
- Le module de placement de lots et d'affichage, utilisé actuellement. Il va récupérer la sortie du logiciel d'optimisation et l'afficher sous forme de proposition de planification. Après exécution de ces décisions, une phase d'actualisation de la base de données permettra au solveur de fonctionner de nouveau avec

le nouvel état de l'atelier.

La majeure partie du travail est d'écrire les programmes qui se chargeront d'assurer ces liaisons et d'appliquer tous ces changements de façon pérenne. Le service informatique s'occupe de cette partie, étant donné sa bonne connaissance des modules impliqués.

La gestion des changements qui en résultent concernant les opérateurs est tout sauf un point de détail. Leur tâche pourrait considérablement changer. Il s'agira d'appliquer des décisions déjà prises et probablement de travailler avec une nouvelle interface. Leur travail risque même parfois d'être amoindri, ce qui pourrait entraîner la redirection vers d'autres activités, ou dans d'autres ateliers. Cette question est un point important du changement associé à cette intégration.

En définitive, la dernière étape du projet soulève des questions préliminaires concrètes. Aucun détail n'est laissé au hasard et il faut voir loin. À ce jour, bien qu'aucune décision n'ait été prise, il est pressenti que dans la suite logique du partenariat entre STMicroelectronics et l'École des Mines de Saint-Étienne, cette dernière s'occupera de la maintenance du logiciel à travers des contrats industriels. Le transfert de connaissances va s'effectuer au cours de trois réunions où plusieurs acteurs concernés s'efforceront de comprendre la nature de la méthode utilisée et les points importants du code source du logiciel. La décision a été prise de changer le moins possible ce code. Un réel engouement s'est manifesté au cours de l'étape d'industrialisation, qui est en bonne voie, ayant démarré en Août 2015 et pas encore achevée en Octobre 2015.

6.5 Conclusion et perspectives

Les travaux menés ont abouti à une réalisation industrielle, ce qui est le but originel de nombreuses études en recherche opérationnelle et aide à la décision. Un projet long, difficile à mener, a été entamé et n'est à ce jour pas terminé. La décision d'industrialisation a été prise et on peut considérer que l'achèvement du projet est en bonne voie. De nombreux enseignements se dégagent de cette expérience et permettent d'éclairer l'idée qu'on a de la mise en œuvre d'un logiciel d'optimisation et d'aide à la décision en milieu industriel.

- La communication et la synchronisation des terminologies permettent d'identifier les éléments importants à prendre en compte dans un modèle d'optimisation et de fluidifier les échanges.
- La validation de l'outil ne peut se faire sans la mise en place d'indicateurs de performance pertinents pour évaluer de façon concrète et objective la qualité d'une solution.
- L'analyse des effets des différentes composantes du problème d'optimisation sur les solutions retournées est une étape primordiale, puisqu'elle permet

d'identifier les éléments superflus du modèle.
- Il n'est pas rare d'entendre évoquer la difficulté engendrée par l'intégration du nouvel outil dans l'adaptation au système d'information et la communication avec diverses bases de données.
- Les aspects de maintenance et de support de l'outil se posent inévitablement. Identifier ce qui est configurable, ce qui peut être exporté, anticiper les obstacles liés à l'implantation informatique en préparant un code portable et suffisamment générique, sont des clés pour accélérer le processus.

Quelques mois séparent alors la décision d'implantation et la réalisation. Plusieurs étapes de transfert de connaissances sont nécessaires, avant l'adaptation du logiciel et la création des interfaces communicantes. Dans un premier temps, des tests a posteriori seront effectués avec l'outil pour déterminer son potentiel gain en situation réelle de fonctionnement. Après diverses périodes d'essai, il sera finalement utilisé, fonctionnant en temps réel et communiquant avec différentes interfaces liées aux bases de données. Ce logiciel déterminera alors un plan de production optimisé à court terme sur des tranches horaires, en considérant toutes les informations actualisées de façon régulière. Plusieurs extensions du projet sont envisagées :

- La migration de l'outil vers d'autres sites de STMicroelectronics. L'entreprise pourrait en utiliser différentes versions, et ainsi procéder à son adaptation. En effet, les unités de production ne fonctionnent pas uniformément et on observe des différences, que le logiciel doit intégrer, d'où une nécessité de préserver une certaine généricité dans le modèle d'ordonnancement en photolithographie. Par exemple, les différences entre les sites de Rousset et de Crolles de STMicroelectronics pourraient à terme se paramétrer dans un seul et même logiciel.
- Une conséquence de ces remarques est l'objectif d'aboutir à une standardisation des systèmes d'information entre différentes unités de production. Une pareille uniformisation faciliterait l'intégration d'outils d'optimisation et d'aide à la décision et serait un réel pas vers l'automatisation.

Chapitre 7

Conclusion générale et perspectives

Le travail réalisé au cours des trois années de thèse s'est soldé par des contributions en ordonnancement sur des problème issus des procédés de fabrication de semi-conducteurs. Des résultats, aussi bien sur la formalisation dans le contexte de l'optimisation combinatoire, sur la complexité théorique des problèmes étudiés, ou les algorithmes de résolution, ont été présentés. Des réalisations industrielles complètent le travail entamé sur l'atelier de photolithographie, notamment un logiciel d'ordonnancement optimisé des lots, mis en œuvre dans une usine d'un grand fabricant de semi-conducteurs.

Il faut rappeler qu'au départ, les enjeux sont énormes d'un point de vue applicatif. La formalisation et l'étude de ce problème trouvent leur origine dans la nécessité grandissante de gérer de façon optimale la production dans un atelier au fonctionnement complexe. La fabrication de semi-conducteurs se distingue par la richesse des processus mis en œuvre et la multiplicité des ressources utilisées. C'est pourquoi l'utilisation d'outils théoriques idoines, notamment issus de la recherche opérationnelle, apparaissent comme une occasion d'améliorer la performance des unités de production en microélectronique. Les travaux de cette thèse se sont donc naturellement axés autour de l'atelier souvent considéré comme le plus critique, celui de photolithographie. Plusieurs études ont révélé que si un gain pouvait se faire en termes de rendement, il s'opérerait en améliorant le fonctionnement de cet atelier.

Pour appliquer des algorithmes, il faut un formalisme décrivant de façon précise la réalité du problème. C'est là que les premiers choix ont été importants. Quels éléments prendre en compte dans le modèle sur lequel l'ensemble des travaux s'effectueront ? Et comment s'insérer le mieux possible dans le cadre déjà bien connu de la théorie de l'ordonnancement ? La tâche est compliquée, au vu des particularités des machines, des produits et de la présence de ressources auxiliaires. Une modélisation s'est finalement construite dans l'optique de coller aux modèles classiques de problèmes d'ordonnancement à machines parallèles, tout en conservant la complexité combinant les contraintes de ressources auxiliaires et celles de temps de setup dépendant de la machine et de la séquence des tâches. Il s'agit donc d'un problème d'ordonnancement à machines parallèles différentes, éligibles, avec ressources auxiliaires unitaires et temps de setup dépendant de la séquence et de la machine. On considère les positions des ressources auxiliaires, avec une position

fictive de stockage et un temps de transport constant égal à 1 lorsqu'une ressource auxiliaire change de position pour l'exécution d'une tâche qui la nécessite. Deux des trois fonctions objectif retenues pour ce problème sont propres à l'application étudiée. En effet, chaque tâche possède un nombre de plaquettes associé et on cherche à maximiser le nombre de plaquettes produites dans une fenêtre de temps fixée. Les tâches ont des priorités associées et la somme pondérée des dates de fin d'exécution des tâches est un critère classique à minimiser. La troisième fonction objectif étudiée, le nombre de transferts de ressources auxiliaires, est à minimiser.

De nombreux résultats sont obtenus pour le problème d'ordonnancement défini dans ce manuscrit :

- Pour le critère de maximisation du nombre de plaquettes traitées dans un horizon de temps fixé, ainsi que pour la minimisation de la somme pondérée des dates de fin d'exécution des tâches, le problème est NP-difficile au sens fort. Cela implique qu'on ne pourra pas trouver d'algorithme de programmation dynamique pseudo-polynomial pour les résoudre dans le cas général.
- Concernant la minimisation du nombre de déplacements de masques, le problème est NP-difficile au sens fort si le nombre de machines est un paramètre du problème et se réduit au problème de *Set Cover*. Si le nombre de machines est constant, le problème devient facile à résoudre, en temps polynomial.
- Des programmes linéaires en nombres entiers sont proposés pour résoudre de manière exacte les trois problèmes. Une formulation commune à variables indexées par le temps est donnée pour le nombre de plaquettes traitées avant un horizon de temps fixé et pour la somme pondérée des dates de fin. Elle présente l'avantage de fournir des bornes de relaxation continues intéressantes mais des modèles mathématiques de taille exponentielle par rapport à la taille des instances.
- Un programme linéaire en nombres entiers, inspiré de la formulation classique pour le problème de couverture par des ensembles (*Set Cover*) permet dans la pratique de résoudre de très grandes tailles d'instances de manière optimale.
- Des inégalités valides pour renforcer les formulations à variables indexées par le temps sont proposées.
- Une heuristique primale permet d'accélérer la détermination d'une solution et d'une borne primale dans l'algorithme de séparation et évaluation.
- Une méthode approchée métaheuristique détermine des solutions pour des instances de grande taille. C'est un algorithme mémétique, à savoir un algorithme génétique hybridé avec une méthode de recherche locale de type recuit simulé. Les atouts de cette méthodes reposent sur :
  - Le codage des solutions dans un formalisme synthétique qui permet de les manipuler facilement au sein de l'algorithme sans perdre la possibilité d'atteindre les solutions optimales. Cela se traduit par le fait que l'ensemble

de solutions accessibles est dominant.
  - Les opérateurs de voisinage assurent la propriété de connexité.
  - Un test permettant d'éviter des voisins inutiles.
  - Une stratégie de convergence de la méthode de recherche locale, basée sur une remise à niveau du paramètre de *température*.
- Une méthode exacte d'optimisation bicritère du problème détermine des solutions Pareto-optimales de façon certaine. Par ailleurs, cette méthode, utilisant une *fonction d'agrégation*, permet d'atteindre la totalité des solutions efficaces (dites Pareto-optimales), selon son paramétrage.

Des poursuites importantes de ces travaux sont envisagées :

- L'étude du polyèdre du problème associé à la formulation à variables indexées par le temps, présentée au chapitre 3, est une extension intéressante. En s'inspirant des travaux de Wolsey et Sousa [80], on peut étudier la dimension du polyèdre et déterminer des inégalités induisant des facettes. Par exemple, Wolsey et Sousa [80] montrent en 1990 que le polyèdre associé à la formulation à variables indexées par le temps pour le problème à une machine est de pleine dimension si l'on relâche les n contraintes d'affectation et si l'on prend la limite des indices temporels T suffisamment grande. On peut conjecturer que le résultat reste le même dans la généralisation étudiée dans cette thèse, c'est-à-dire avec des machines parallèles différentes, des temps de setup dépendant de la machine et de la séquence, ainsi qu'avec des contraintes d'éligibilité de machines et de ressources auxiliaires.
- L'algorithme mémétique présenté au chapitre 4 possède d'intéressantes propriétés, en particulier les opérateurs de voisinage utilisés dans la méthode de recherche locale. Avec l'ajout de la notion de *voisins significatifs*, on évite, par une vérification préalable, de complexité pire-cas $O(n)$, des voisinages qui ne modifient pas la solution. Avec cet ajout, on conjecture que la méthode de recherche locale conserve son caractère *connexe*, au sens où toute solution est accessible à partir d'une solution initiale donnée.
- La façon de paramétrer la norme de Tchebychev pondérée augmentée, mieux étudiée, pourrait aboutir à une génération précise du front de Pareto dans certains cas.
- La méthode fournie pour la minimisation du nombre de déplacements de masques, du fait du caractère non régulier de ce critère, offre des degrés de liberté dans la solution. En d'autres termes, le nombre de transferts de masques n'est impacté que par un sous-ensemble des variables de décision, laissant la liberté de fixer les autres. Nous pourrions ainsi développer une méthode à deux phases, tirant parti de l'optimum sur ce critère. Si on considère l'espace des critères, elle permettrait, de rechercher dans l'hyperplan formé par les points ayant cette valeur pour troisième composante, la so-

lution qui optimise une fonction d'agrégation bicritère sur les deux autres objectifs considérés. Cette méthode garantirait de retourner des solutions Pareto-optimales tricritère.

Concernant les aspects industriels, plusieurs enseignements et conclusions se dégagent de la thèse :

- Dans toute collaboration, la communication est importante. Deux mondes cohabitent, les langages, discours et terminologies diffèrent. Il faut écouter d'une oreille différente, appréhender les contradictions internes, savoir traduire des souhaits en objectifs, des requis en contraintes. Cette acuité se travaille, et pour parvenir à distinguer le nécessaire du superflu, rien n'est plus efficace que la connaissance des systèmes et procédures. Il est crucial de se documenter auprès des services compétents et aguerris, sur les machines et les produits concernés. Seule l'expérience de ces procédés permet de prédire avec précision quels éléments impactent la production, et sont donc à prendre en compte dans un modèle d'optimisation et d'aide à la décision.
- La validation est une question délicate. Pour convaincre, il a fallu faire des tests qui répondent aux exigences de l'entreprise. Parfois, les indicateurs retenus diffèrent selon l'interlocuteur. Il faut une méthode suffisamment robuste pour produire des solutions efficaces au sens de différents objectifs, très souvent corrélés. Dans notre cas, il a fallu une longue période pour parvenir à la conclusion commune que certains critères évoqués au cours des réunions, bien que différents, étaient dans les cas pratiques traités, équivalents au sens où ils produisaient des solutions similaires.
- Le degré de détail à apporter à la modélisation du problème peut être un frein important au développement d'un logiciel. En l'occurence, en dépit des solutions cohérentes que retournait l'algorithme, certains de nos interlocuteurs voulaient augmenter le niveau de détail, notamment dans l'enchaînement des lots sur les machines, jugeant le modèle basé sur le calcul des durées d'exécution (*P-Time, C-Time*) quelque peu approximatif. De nombreux tests et simulations nous ont finalement fait converger vers un modèle suffisamment précis sans s'encombrer de contraintes dépourvues d'effet majeur sur l'ordonnancement. Cette discussion est très importante car elle permet de rendre compte de la réelle perception du rôle d'un outil d'optimisation et d'aide à la décision dans l'industrie. Il ne faut certes pas négliger l'importance d'une modélisation précise et adaptée, mais il est trompeur de le voir comme un oracle qu'on consulterait pour anticiper sur le futur avec certitude. Une illustration de ce fait est notamment la différence entre les notions d'*ordonnancement* et de *séquence*. Un algorithme d'optimisation propose avant tout une séquence dans ce type d'applications. Les dates de début et de fin (qui en font un ordonnancement) n'ont qu'une valeur indicative et la précision de ces données

dépend du degré de détail intégré au modèle.

- Lorsque les effets d'une intégration logicielle sont étendus et bouleversent le fonctionnement de tout un mode de fonctionnement, comme c'est le cas dans ce projet, il est encore plus difficile d'obtenir l'aval des décideurs. Le résultat positif est d'autant plus réjouissant. Durant le projet, les conséquences de cette intégration se sont dessinées progressivement. On se rend compte que ce logiciel s'inscrit dans le cadre d'une automatisation de l'atelier de photolithographie. Il s'agit là de poser quelques bases d'un grand édifice, et d'initier un travail de longue haleine.
- À l'heure de poursuivre dans l'introduction de l'outil dans l'atelier, plusieurs questions essentielles se posent, au premier rang desquelles le support et la maintenance du produit. Mais on s'aperçoit également qu'une anticipation pertinente des difficultés liées à l'implantation permet un gain de temps non négligeable. Pour l'essentiel, il s'agit de la cohérence entre plate-formes (donc préparer un code portable, ce qui a été le cas), l'écriture d'un code lisible et réutilisable (bien commenté et respectant des conventions d'écriture communément admises), et répondre à des exigences d'efficacité (code optimisé, par exemple).

La suite logique de ces accomplissements industriels serait d'étendre ces travaux à d'autres sites de STMicroelectronics. Un enjeu majeur d'une telle extension serait d'arriver à une uniformisation des protocoles (extraction de données de l'atelier de photolithographie) en mettant en place un outil suffisamment générique pour traiter les spécificités des sites de production.

Bibliographie

[1] E. Akçali, K. Nemoto et R. Uszoy – « Cycle-time improvements for photolithography process in semiconductor manufacturing », *IEEE Transactions on Semiconductor Manufacturing* **14** (2001), no. 1, p. 48–56.

[2] M. van den Akker, C. Hurkens et M. Savelsbergh – « Time-Indexed Formulations for Machine Scheduling Problems : Column Generation », *INFORMS Journal on Computing* **12** (2000), no. 2, p. 111–124.

[3] A. Allahverdi, C. T. Ng, C. T. E. Cheng et M. Y. Kovalyov – « A survey of scheduling problems with setup times or costs », *European Journal of Operational Research* **187** (2008), no. 3, p. 985–1032.

[4] L. Atherton et R. Atherton – *Wafer fabrication : Factory performance and analysis*, Kluwer Academic Publishers, 1995.

[5] G. Beliakov, A. Pradera et T. Calvo – *Aggregation functions : A guide for practitioners*, Studies in Fuzziness and Soft Computing, Springer, 2007.

[6] J. Blazewicz, K. Ecker, E. Pesch, G. Schmidt et J. Weglarz – *Handbook on scheduling*, Springer, 2007.

[7] J. Blazewicz, J. Lenstra et A. R. Kan – « Scheduling subject to resource constraints : Classification and complexity », *Discrete Applied Mathematics* **5** (1983), no. 1, p. 11–24.

[8] S. Bouveret et M. Lemaître – « Computing leximin-optimal solutions in constraint networks », *Artificial Intelligence* **173** (2009), no. 2, p. 343–364.

[9] P. Brucker, B. Jurisch et A. Kramer – « Complexity of scheduling problems with multi-purpose machines », *Annals of Operations Research* **70** (1997), no. 0, p. 57–73.

[10] J. Bruno, E. Coffman et M. Sethi – « Scheduling independent tasks to reduce mean finishing time », *Journal of the ACM* **17** (1974), no. 7, p. 382–387.

[11] P. Camerini, G. Galbiati et M. Maffioli – « Complexity of spanning tree problems : Part i », *European Journal of Op. Res.* **5** (1980), p. 346–352.

[12] S. Chand et H. Schneeberger – « Single machine scheduling to minimize weighted completion time with maximum allowable tardiness », Research report, University of Purdue, 1984.

[13] — , « A note on the single machine scheduling problem with minimum weighted completion time and maximum allowable tardiness », *Naval Res. Logist. Quarterly* **33** (1986), p. 551–557.

[14] G. Choquet – « Théorie des capacités », *Ann. Institut Fourier (Grenoble)* (1953), p. –.

[15] V. Chvàtal – « A greedy heuristic for the set covering problem », *Math. Oper. Res.* **4** (1979), no. 3, p. 233–235.

[16] R. Conway, W. Maxwell et L. Miller – *Theory of Scheduling*, Courier Corporation, 1967.

[17] S. Dauzère-Pérès et M. Sevaux – « An exact method to minimize the number of tardy jobs in single machine scheduling », *Journal of Scheduling* **7** (2004), no. 6, p. 405–420.

[18] F. Della Croce et V. T'Kindt – « Improving the preemptive bound for the single machine min-max lateness problem subject to release times », *Operations Research Letters* **38** (2010), no. 10, p. 589–591.

[19] M. Dyer et L. Wolsey – « Formulating the single machine sequencing problem with release dates as a mixed integer program », *Discrete Applied Mathematics* **26** (1990), no. 2-3, p. 255–270.

[20] M. Ehrgott et X. Gandibleux – « A survey and annotated bibliography of multiobjective combinatorial optimization », *OR Spectrum* **22** (2000), p. 425–460.

[21] I. T. on Fundamentals of Elec. Comm. et C. Sc. (éds.) – *A genetic approach for maximum independent set problems*, 1997.

[22] V. Emelichev et V. Perepelitsa – « On the cardinality of the set of alternatives in discrete many-criterion problems », *Discrete Mathematics and Applications* **2** (1992), no. 5, p. 461–471.

[23] H. Emmons – « A note on a scheduling problem with dual criteria », *Naval Res. Logist. Quarterly* **22** (1975), p. 615–616.

[24] B. Estève, C. Aubijoux, A. Chartier et V. T'kindt – « A recovering beam search algorithm for the single machine just-in-time scheduling problem », *European Journal of Operational Research* **172** (2006), no. 3, p. 798–813.

[25] M. Garey et D. Johnson – « Strong np-completeness results : motivation, examples, and implications », *J. Assoc. Comput. Mach.* **3** (1978), no. 25, p. 499–508.

[26] F. Gembicki – « Vector optimization for control with performance and parameter sensitivity indices », Ph.D. Thesis, Case Western Reserve University, 1973.

[27] M. Gendreau, G. Laporte et E. M. Guimaraes – « A divide and merge heuristic for the multiprocessor scheduling problem with sequence dependent

setup times », *European Journal of Operational Research* **133** (2001), p. 183–189.

[28] M. Grabisch, J. Marichal, R. Mesiar et E. Pap – « Aggregation functions », Cambridge University Press, 2009.

[29] R. Graham – « Bounds for certain multiprocessing anomalies », *Bell System Tech. J.* **45** (1966), p. 1563–1581.

[30] — , « Bounds on multiprocessing timing anomalies », *SIAM Journal of Applied Mathematics* **17** (1969), p. 263–269.

[31] A. Grigoriev, M. Sviridenko et M. Uetz – « Unrelated parallel machine scheduling with resource dependent processing times », *Integer Programming and Combinatorial Optimization, Lecture Notes in Computer Science* **3509** (2005), p. 182–195.

[32] J. Gupta, R. Ruiz, J. Fowler et S. Mason – « Operational planning and control of semiconductor wafer production », *Production Planning and Control : The Management of Operations* **17** (2006), no. 7, p. 639–647.

[33] Y. Haimes, L. Ladson et D. Wismer – « On a bicriterion formulation of the problems of integrated system identification and system optimization », *IEEE Trans. Systems, Man and Cybernetics* **1** (1971), p. 296–297.

[34] L. Hall, S. Schulz et D. Shmoys – « Scheduling to minimize average completion times : Off-line and on-line algorithms », *Proceedings of 17th ACM-SIAM Symposium on Discrete Algorithms*, 1996, p. 142–151.

[35] S. Hartmann – « A competitive genetic algorithm for resource-constrained project scheduling », *Naval Research Logistics* **45** (1998), p. 733–750.

[36] J. Holland – *Adaptation in natural and artificial systems*, University of Michigan Press, 1975.

[37] J. Hoogeveen – « Single-machine bicriteria scheduling », Ph.D. Thesis. CWI Amsterdam, 1992.

[38] O. Ibarra et C. Kim – « Heuristic Algorithms for Scheduling Independent Tasks on Nonidentical Processors », *Journal of the ACM* **24** (1977), no. 2, p. 280–289.

[39] T. John – « Tradeoff solutions in single machine production scheduling for minimizing flow time and maximum penalty », *Comput. Oper. Res.* **16** (1984), p. 471–479.

[40] D. Johnson – « Approximation Algorithms for Combinatorial Problems », *Journal of Computer Systems Science* **9** (1974), no. 3, p. 256–278.

[41] D. Kalyanmoy et K. Miettinen – « Nadir point estimation procedures using evolutionary approaches : Better accuracy and computational speed

through focused search », *Multiple Criteria Decision Making for Sustainable Energy and Transportation*, Springer, 2008, p. 339–354.

[42] R. Karp – « Reductibility among combinatorial problems », Complexity of Computer Computations (R. E. Miller et J. W. Thatcher, éds.), Plenum Press, 1972, p. 85–104.

[43] T. Kawaguchi et S. Kyan – « Worst case bound of an lrf schedule for the mean weighted flow-time problem », *SIAM Journal on Computing* **15** (1986), no. 4, p. 1119–1129.

[44] C. Koulamas et G. J. Kyparisis – « Single-machine scheduling problems with past-sequence-dependent setup times », *European Journal of Operational Research* **187** (2008), no. 3, p. 1045–1049.

[45] E. Lawler – « Sequencing jobs to minimize total weighted completion time subject to precedence constraints », *Annals of Discrete Mathematics* **2** (1978), p. 75–90.

[46] E. Lawler et J. Moore – « A Functional Equation and its Application to Resource Allocation and Sequencing Problems », *Management Science* **16** (1969), no. 1, p. 77–84.

[47] C. Lee et R. Uzsoy – « A new dynamic programming algorithm for the parallel machines total weighted completion time problem », *Operations Research Letters* **11** (1992), no. 2, p. 73–75.

[48] J. Lenstra – Non publié.

[49] J. Lenstra et A. Rinnooy Kan – « Complexity of scheduling under precedence constraints », *Operations Research* **26** (1978), p. 22–35.

[50] J. Lenstra, A. Rinnooy Kan et P. Brucker – « Complexity of machine scheduling problems », *Annals of Discrete Mathematics* **1** (1977), p. 343–362.

[51] J. Lenstra, D. Shmoys et E. Tardos – « Approximation Algorithms for Scheduling Unrelated Parallel Machines », *Mathematical Programming* **46** (1990), p. 259–271.

[52] J. Leung et G. Young – « Minimizing schedule length subject to minimum flow time », *SIAM Jounal of Comput.* **18** (1989), p. 314–326.

[53] J. Leung et C. Li – « Scheduling with processing set restrictions : A survey », *International Journal of Production Economics* **116** (2008), no. 2, p. 251–262.

[54] L. Lovász – « On the ratio of the optimal integral and fractional covers », *Discrete Mathematics* **13** (1975), no. 4, p. 383–390.

[55] M. Luque, K. Miettinen, P. Eskelinen et F. Ruiz – « Incorporating preference information in interactive reference point methods for multiobjective optimization », *Omega* **37** (2009), p. 450–462.

[56] M. Luque, F. Ruiz et K. Miettinen – « Global formulation for interactive multiobjective optimization », *OR Spectrum* **33** (2011), no. 1, p. 27–48.

[57] S. McCormick et M. Pinedo – « Scheduling n independent jobs on m uniform machines with both flowtime and makespan objectives : A parametric analysis », *ORSA J. Comput.* **7** (1995), p. 63–77.

[58] A. S. Mendes, F. M. Muller, P. M. França et P. Moscato – « Comparing meta-heuristic approaches for parallel machine scheduling problems », *Production Planning and Control* **13** (2002), p. 143–154.

[59] P. Merz et B. Freisleben – « Fitness landscapes, memetic algorithms and greedy operators for graph bi-partitioning », *Evolutionary Computation* **1** (2000), no. 8, p. 61–91.

[60] R. M'Hallah et R. Bulfin – « Minimizing the weighted number of tardy jobs on a single machine with release dates », *European Journal of Operational Research* **176** (2007), no. 2, p. 727–744.

[61] L. Min et W. Cheng – « Identical parallel machine scheduling problem for minimizing the makespan using genetic algorithm combined with simulated annealing », *Chinese Journal of Electronics* **7** (1998), no. 4, p. 317–321.

[62] S. Miyazaki – « One machine scheduling problem with dual criteria », *J. Oper. Res. Soc. Jpn* **24** (1981), p. 37–50.

[63] L. Mönch, J. Fowler et S. Mason – *Production planning and control for semiconductor wafer fabrication facilities*, Operations Research/Computer Science Interfaces Series, vol. 52, Springer, 2013.

[64] L. Moalic et A. Gondran – « The new memetic algorithm head for graph coloring : An easy way for managing diversity », p. 173–183, Springer, 2015.

[65] C. Monma et C. Potts – « On the complexity of scheduling with batch setup times », *Operations Research* **37** (1989), no. 5, p. 798–804.

[66] P. Moscato et C. Cotta – « A modern introduction to memetic algorithms », ch. 6, p. 141–184, Springer, 2010.

[67] T. Murofushi et M. Sugeno – « Fuzzy measures and fuzzy integrals », (M. Grabisch, T. Murofushi et M. Sugeno, éds.), Physica-Verlag, 2000, p. 3–41.

[68] C. Nash – « Origins of the Simplex Method », p. 141–151, ACM Press Hist. Ser., 1990.

[69] Y. Nikulin, K. Miettinen et M. Mäkelä – « A new achievement scalarizing function based on parametrization in multiobjective optimization », *OR Spectrum* **34** (2012), no. 1, p. 69–87.

[70] A. Obeid, S. Dauzère-Pérès et C. Yugma – « Scheduling job families on non-identical parallel machines with time constraints », *Annals of Operations Research* **213** (2014), no. 1, p. 221–234.

[71] V. Pareto – « Manuel d'économie politique », Qiard, Paris, 1909.

[72] G. Rabadi, R. Moraga et A. Al-Salem – « Heuristics for the unrelated parallel machine scheduling problem with setup times », *Journal of Intelligent Manufacturing* **17** (2006), no. 1, p. 85–97.

[73] C. Reeves – « Genetic algorithms », ch. 5, p. 109–140, Springer, 2010.

[74] M. Rothkopf – « Scheduling independent tasks on parallel processors », *Management Science* **12** (1966), p. 347–447.

[75] S. Sahni – « Algorithms for scheduling independent tasks », *Journal of the ACM* **23** (1976), no. 1, p. 116–127.

[76] Y. Sawaragi, H. Nakayama et T. Tanino – *Theory of multiobjective optimization*, Academic Press, Orlando, 1985.

[77] P. Serafini – « Some considerations about computational complexity for multi objective combinatorial problems », (J. Jahn et W. Krabs, éds.), vol. 294, Springer, 1986.

[78] E. Shchepin et N. Vakhania – « An optimal rounding gives a better approximation for scheduling unrelated machines », *Operations Research Letters* **33** (2005), no. 2, p. 127–133.

[79] W. Smith – « Various optimizers for single-stage production », *Naval Research Logistics Quarterly* **3** (1956), no. 1, p. 59–66.

[80] J. Sousa et L. Wolsey – « A time indexed formulation of non-preemptive single machine scheduling problems », *Mathematical Programming* **54** (1992), p. 353–367.

[81] I. E. M. T. Symposium (éd.) – *A sequential solution methodology for capacity allocation and lot scheduling problems for photolithography.*, 2000.

[82] V. T'Kindt et J. Billaut – *Multicriteria scheduling : Theory, models and algorithms*, vol. 1, Springer, 2006.

[83] V. T'Kindt, F. Della Croce et J. Bouquard – « Enumeration of pareto optima for a flowshop scheduling problem with two criteria », *INFORMS Journal on Computing* **19** (2007), no. 1, p. 64–72.

[84] R. Uszoy, C. Lee et L. Martin-Vega – « A review of production planning and scheduling models in the semiconductor industry-part i : system characteristics, performance evaluation and production planning », *IIE Transactions* **24** (1992), no. 4, p. 47–60.

[85] G. Vilcot, J. Billaut et C. Esswein – « A genetic algorithm for a bicriteria flexible job shop problem », *European Journal of Operational Research* **190** (2008), no. 2, p. 398–411.

[86] S. Webster et M. Azizoglu – « Dynamic programming algorithms for scheduling parallel machines with family setup times », *Computers & Operations Research* **28** (2001), no. 2, p. 127–137.

[87] M. X. Weng, J. Lu et H. Ren – « Unrelated parallel machine scheduling with setup consideration and a total weighted completion time objective », *International Journal of Production Economics* **70** (2001), p. 215–226.

[88] D. White – « A special multi-objective assignment problem », *Journal of The Operational Research Society* **8** (1984), p. 759–767.

[89] A. Wierzbicki – « The use of reference objectives in multiobjective optimization », *MCDM theory and Application, Proceedings* (1980), no. 177, p. 468–486.

[90] — , « On the completeness and constructiveness of parametric characterizations to vector optimization problems », *OR Spectrum* **8** (1986), p. 73–87.

[91] G. Woeginger et M. Skutella – « A ptas for minimizing the total weighted completion time on identical parallel machines », *Mathematics of Operations Research* **25** (2000), no. 1, p. 63–75.

[92] R. R. Yager – « On ordered weighted averaging aggregation operators in multicriteria decision making », *IEEE Trans. Systems, Man and Cybern* (1988), no. 18, p. 183–190.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT: *2015 EMSE 0808*

Abdoul Menhem BITAR

DISSERTATION TITLE: PARALLEL MACHINE SCHEDULING FOR SEMICONDUCTOR MANUFACTURING: PHOTOLITHOGRAPHY WORKSTATIONS

Speciality: Industrial Engineering

Keywords: Scheduling, Combinatorial optimization, Integer Linear Programming, Memetic algorithm, Multicriteria

Abstract:

Semiconductor manufacturing has grown considerably in recent decades, due to new industrial applications of microelectronic devices. The related manufacturing process is known to be complex. A bottleneck process step, the photolithography workshop, gathers various types of constraints, related to the number of auxiliary resources and the tools characteristics. The aims of the thesis were to model this workstation as a parallel machine scheduling problem and to optimize various criteria, determined by industrial needs. Some complexity results are provided and optimization algorithms led to an industrial application and a software that provides optimized schedules in a specific fab.

École Nationale Supérieure des Mines
de Saint-Étienne

NNT : *2015EMSE 0808*

Abdoul Menhem BITAR

TITRE DE LA THESE : ORDONNANCEMENT SUR MACHINES PARALLELES APPLIQUE A LA FABRICATION DE SEMICONDUCTEURS : ATELIERS DE PHOTOLITHOGRAPHIE

Spécialité: Génie Industriel

Mots clefs : Ordonnancement, Optimisation combinatoire, Programmation Linéaire en Nombres Entiers, Algorithme mémétique, Multicritère

Résumé :

Le secteur des semi-conducteurs a connu un développement considérable ces dernières décennies, du fait des nouvelles applications de la microélectronique dans l'industrie. Le processus de fabrication est réputé pour sa complexité. L'un des ateliers les plus critiques de la production, l'atelier de photolithographie, est régi par un ensemble conséquent de contraintes de production. La multiplicité des ressources utilisées, le nombre important de produits traités, en font une zone importante à optimiser. Les objectifs de la thèse ont été de modéliser cet atelier sous la forme d'un problème d'ordonnancement sur machines parallèles et d'optimiser plusieurs critères jugés pertinents pour évaluer la qualité des solutions. Des résultats en termes de complexité, et d'algorithmes de résolution, ont permis une application industrielle, dans la mesure où un logiciel d'optimisation destiné à l'ordonnancement des lots en photolithographie a été développé.